

10th National Bureau of Standards/National Computer Security Center
National Computer Security Conference

Proceedings



21-24 September 1987

***"COMPUTER SECURITY...
from principles to practices."***

WELCOME

The National Computer Security Center and the Institute for Computer Sciences and Technology are pleased to welcome you to the Tenth Annual National Computer Security Conference. The past nine conferences have stimulated the sharing of information and the application of this new technology. We are confident this year's conference will continue this tradition.

This year's conference theme -- Computer Security: From Principles to Practices -- reflects the growth of computer security awareness and a maturation of the technology of trusted systems. Our next major challenge is to understand how to build secure applications on these trusted bases. The efforts of the National Computer Security Center, the Institute for Computer Sciences and Technology, computer users, and the computer industry have all contributed to the advances in computer security over the past few years. We are committed to a vibrant partnership between the Federal Government and private industry in furthering the state of the art in Computer Security.

The great challenge of the future is for us to build upon the bases we are now developing so that new applications can emerge. We must understand and "record" how we build on these foundations in order to secure end products. To be successful, we need your help as you take back to your places of work an increased awareness of where we are and where we must go.


JAMES H. BURROWS
Director

*Institute for Computer Sciences
and Technology*


PATRICK R. GALLAGHER, JR.
Director

National Computer Security Center

TABLE OF CONTENTS

Network Security

- 1 *Developments in Guidance for Trusted Networks*
Alfred W. Arsenault, National Computer Security Center
- 9 *Considerations for Security in the OSI Architecture*
Dennis K. Branstad, National Bureau of Standards
- 15 *A Mission-Critical Approach to Network Security*
Howard L. Johnson, Information Intelligence
Sciences, Inc.
J. Daniel Layne, Computer Technology Associates,
Inc.

Network and Distributed Systems

- 25 *A Security Policy and Model for a MLS LAN*
Peter Loscocco, National Computer Security Center
- 38 *Security in Open Systems -- A Report on the Standards
Work of ECMA's TC32/TG9*
T. A. Parker, ICL Defence Systems, UK
- 51 *Applying the Orange Book to an MLS LAN*
Dan Schnackenberg, Boeing Aerospace Company
- 56 *Information Flow Control in a Distributed Object-
Oriented System with Statically Bound Object
Variables*
Arthur E. Oldehoeft and Masaaki Mizuno, Iowa State
University
- 68 *The Architecture of a Distributed Trusted Computing
Base*
Jon Fellows and Judy Hemenway, Nancy Kelem,
Sandra Romero, UNISYS
- 78 *Specification and Verification Tools for Secure
Distributed Systems*
J. Daniel Halpern and Sam Owre, Sytek Inc.

Management Practices

- 84 *Specification for a Canonical Configuration Accounting
Tool*
R. Leonard Brown, Aerospace Corporation

- 91 *RACF Implementation at Puget Power*
Arturo Maria, Puget Power
- 98 *Management Actions for Improving DoD Computer Security*
William Neugent, MITRE Corporation

Risk Management

- 103 *Risk Analysis and Management in Practice for the UK Government -- the CCTA Risk Analysis and Management Methodology: CRAMM*
Robin H. Moses, UK CCTA; E. Rodney Clark, BIS Applied Systems, LTD
- 108 *A Plan for the Future*
Sylvan Pinsky, National Computer Security Center

Verification Theory

- 109 *m-EVES*
Dan Craigen, I. P. Sharp Associates Limited
- 118 *The Bell-LaPadula Computer Security Model Represented as a Special Case of the Harrison-Ruzzo-Ullman Model*
Paul A. Pittelli, Department of Defense
- 122 *Comparing Specification Paradigms for Secure Systems: Gypsy and the Boyer-Moore Logic*
Matt Kaufmann, William D. Young, University of Texas

Architectural Issues

- 129 *Locking Computers Securely*
O. Sami Saydjari, Joseph M. Beckman,
Jeffrey R. Leaman, National Computer Security Center
- 142 *UNIX and B2 - Are They Compatible?*
W. Olin Sibert, Oxford Systems, Inc.;
Deborah D. Downs, Kenneth B. Elliott III, Aerospace Corporation; Jeffrey J. Glass, MITRE Corporation;
Holly M. Traxler, Grant M. Wagner, National Computer Security Center

SDNS: A Network on Implementation

- 150 *The Secure Data Network System: An Overview*
Gary L. Tater, Edmund G. Kerut
- 153 *SDNS Services and Architecture*
Ruth Nelson, GTE Government Systems Corporation

- 158 *SP-4: A Transport Encapsulation Security Protocol*
Dennis Branstad, National Bureau of Standards
Joy Dorman, Digital Equipment Corporation
Russell Housley, Xerox Corporation
James Randall, International Business Machines Corporation
- 162 *SDNS Products in the Type II Environment*
John Linn, BBN Communications Corporation
- 165 *Access Control Within SDNS*
Edward R. Sheehan, Analytics Incorporated
- 172 *An Overview of the Caneware Program*
Herbert L. Rogers, National Security Agency

Modeling and Verification Tools

- 175 *Ina Flo: The FDM Flow Tool*
Steven T. Eckmann, Unisys Corporation
- 183 *A Gypsy Verifier's Assistant*
Ben L. Di Vito, Larry A. Johnson, TRW Defense Systems Group
- 193 *Formal Models, Bell and LaPadula and Gypsy*
Tad Taylor, Bret Hartman, Research Triangle Institute

Vendor Activities

- 201 *TRUDATA: The Road to a Trusted DBMS*
Ronald B. Knode, ORI/INTERCON Systems Corporation
- 211 *The Sybase Secure Dataserver: A Solution to the Multilevel Secure DBMS Problems*
Patricia A. Rougeau, Edward D. Sturms, TRW Federal Systems Group
- 216 *Computer Security at Sun Microsystems, Inc.*
Katherine Addison, Larry Baron, Don Cragun,
Mark Copple, Keith Hospers,
Patricia Jordan, Mikel Lechner, Michael Manley,
Casey Schaufler, Sun Microsystems Federal, Inc.

Specific Threats

- 220 *Taxonomy of Computer Virus Defense Mechanisms*
Catherine L. Young, National Computer Security Center
- 226 *Computer Viruses: Myth or Reality?*
Howard Israel, National Computer Security Center

- 231 *What Do You Feed a Trojan Horse?*
Dr. Cliff Stoll, Lawrence Berkeley Laboratory
- 238 *Towards the Elimination of the Effects of Malicious Logic: Fault Tolerance Approaches*
Mark K. Joseph, University of California,
Los Angeles

Security in UNIX

- 245 *The Setuid Feature in UNIX and Security*
Steve Bunch, Gould Computer Systems Division
- 254 *Networking of Secure Xenix Systems*
Wilhelm Burger, IBM Corporation Federal Systems
Division
- 257 *A Least Privilege Mechanism for UNIX*
Frank Knowles, Steve Bunch, Gould Computer
Systems Division

DoD Computer Security R&D Programs

- 263 *An Overview of the DoD Computer Security Research and Development Program*
Larry Castro, National Computer Security Center

Evaluation and Certification

- 266 *Certification: A Risky Business*
Martin Ferris, Andrea Cerulli, Department of the
Treasury
- 273 *Security Evaluations of Computer Systems*
David J. Lanenga, National Computer Security Center
- 277 *An Expert System Approach to Security Inspection of a VAX/VMS System in a Network Environment*
Henry S. Teng, Digital Equipment Corporation
Dr. David C. Brown, Worcester Polytechnic Institute
- 282 *The Application "Orange Book" Standards to Secure Telephone Switching Systems*
Capt. Paul D. Engelman, HQ AFCC/AIZ
- 288 *The National Computer Security Center Technical Guidelines Program*
Phillip H. Taylor, National Computer Security Center

Training and Awareness

- 298 *Getting Organizations Involved in Computer Security:
The Role of Security Awareness*
Elizabeth Markey, U.S. Department of State
- 300 *The Computer Security Training Base of 1985*
Eliot Sohmer, National Computer Security Center
- 316 *Department of the Navy Automated Data Processing
Security Program Training OPNAVINST 5239.1A*
Patricia A. Grandy, Navy Regional Data Automation
Center

Social Issues and Ethics

- 320 *Social Aspects of Computer Security*
Dorothy E. Denning, Peter G. Neumann, Donn B. Parker
SRI International
- 326 *Security and Privacy: Issues of Professional Ethics*
Marlene Campbell, Murray State University

Data Base Management Security

- 334 *Data Integrity vs. Data Security: A Workable
Compromise*
Ronda R. Henning, Swen A. Walker, National Computer
Security Center
- 340 *Status of Trusted Database Management System
Interpretations*
Michael W. Hale, National Computer Security Center

The Insider Threat

- 343 *Insider Threat Identification Systems*
Allan R. Clyde, A. R. Clyde Associates

ADA Verification Issues

- 357 *ADA Technology/COMPUSEC Insertion Status Report*
Kenneth E. Rowe, Clarence O. Ferguson, Jr.,
National Computer Security Center
- 362 *The Use of ADA in Secure and Reliable Software*
Mark E. Woodcock, National Computer Security Center

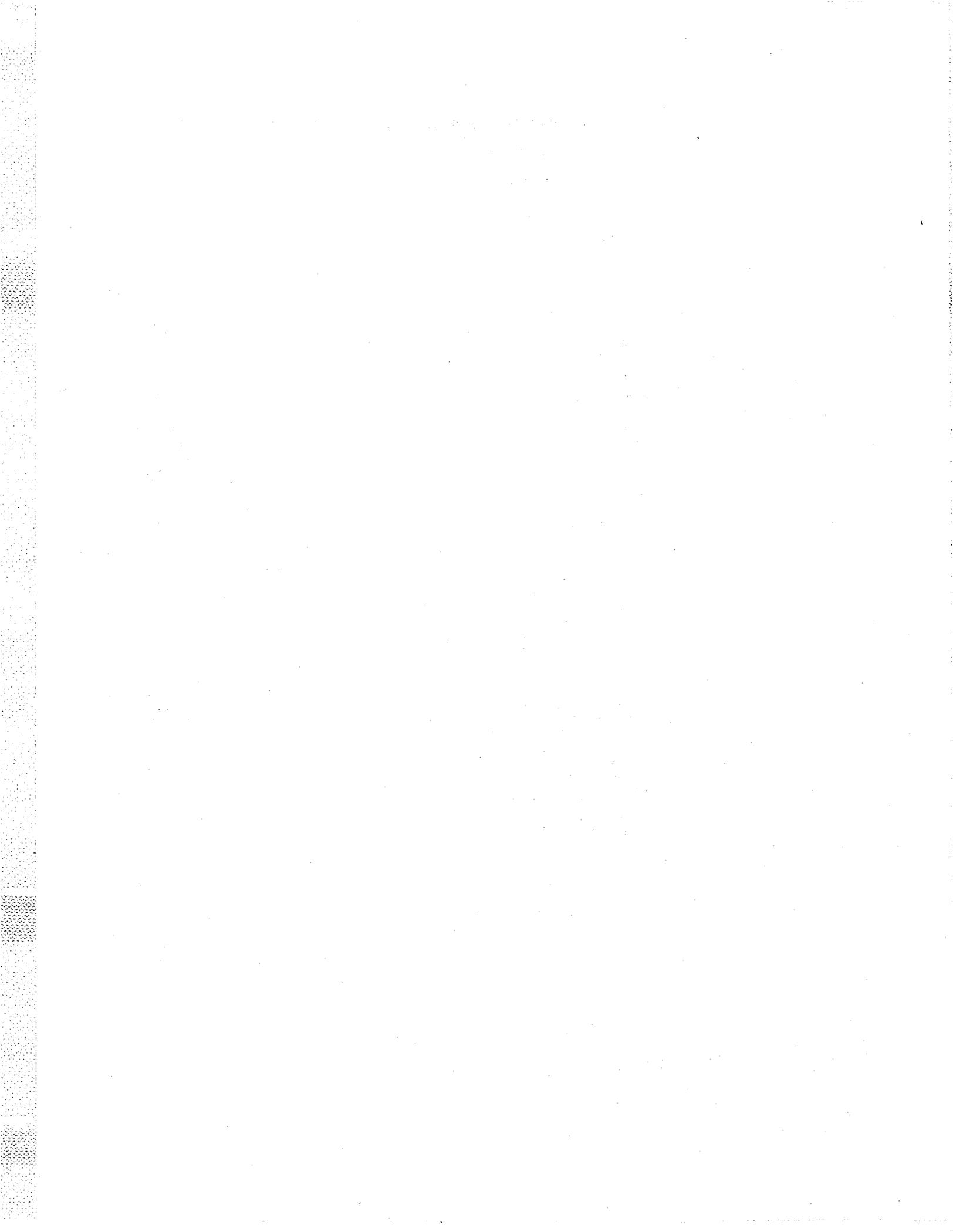
- 366 *An ADA Verification Environment*
 David Guaspari, C. Douglas Harper,
 Norman Ramsey, Odyssey Research Associates

Contingency Planning

- 373 *Computer Disaster Recovery Planning: A Fast Track Approach*
 O. R. Pardo, Bechtel Eastern Power Corporation
- 379 *Return to Normalcy: Issues in Contingency Processing*
 Thomas C. Judd, Federal Reserve System Contingency
 Processing Center
 Howard W. Ward, Germanna Community College

Small Systems Management

- 384 *Advisory Memorandum on Office Automation Security: An Overview*
 Alfred W. Arsenault, National Computer Security Center



DEVELOPMENTS IN GUIDANCE FOR TRUSTED COMPUTER NETWORKS

Alfred W. Arsenault

National Computer Security Center
Ft. George G. Meade, MD

Abstract

The Technical Guidelines Division of the NCSC has been working to produce guidance for Trusted Computer Networks that would be analogous to that provided for stand-alone computer systems by the Trusted Computer System Evaluation Criteria. This paper discusses the latest events in that development: the Trusted Network Interpretation (TNI), how the TNI came to be, what its implications are, and what lies ahead.

Introduction

The purpose of this paper is to discuss the current status and future plans for guidance in the area of trusted computer networks. The National Computer Security Center ("the Center") has been working on this problem since late 1983; earlier stages in the development can be seen in the proceedings of the New Orleans Workshop [1] and in the draft Trusted Network Evaluation Criteria [2], or "Brown Book". In April of 1987, the Center distributed for review the draft Trusted Network Interpretation [3], or "TNI".

The New Philosophy

Comments received on the Brown Book led the Center to believe that it did not reflect the right approach to network security. Therefore, it was necessary to reexamine some of the early results, and to take a different approach to developing network guidance. This new approach is actually a marriage of some of the early recommendations. It involves the realization that, although not all networks can be evaluated and assigned a single rating, some can. Specifically, the working group responsible for producing the TNI believes that the reference monitor concept¹ is appropriate for certain network systems. These systems fit what is called the "single trusted system view" - that is, they can accurately be regarded as an instance of a

single trusted system. Networks of this type have a single trusted computing base, referred to as the Network Trusted Computing Base (NTCB). The NTCB is partitioned among the network components in a manner that ensures the overall network security policy is enforced by the network as a whole.

The implication of this is that these networks can be evaluated, using the concepts embodied in the Trusted Computer System Evaluation Criteria [4] (TCSEC) as the basis for the evaluation. The words in the TCSEC may not apply directly; they must be interpreted as necessary for the network context. Additionally, these requirements may need to be augmented by other requirements, such as those for "other security services" like Communications Integrity, Authentication, Non-Repudiation, and Assurance of Service. However, it is important to realize that the fundamental concepts of network evaluations are those described in the TCSEC; new concepts are introduced only where essential to understand the TCSEC in the network context.

Networks for which no meaningful evaluation is possible are addressed using the "interconnected accredited Automated Information System (AIS) view."² The interconnected accredited AIS view is an operational perspective that recognizes that parts of the network may be independently created, managed, and accredited. Each AIS is accredited to handle sensitive information at a single level or over a range of levels. In this view, the individual AIS may be thought of as "devices" with which neighboring components can send and receive information. An interconnection rule must be enforced to limit the levels of information communicated across the network.

The difference between these two views is simple, and it is a major one. When a "single trusted system" (or a component, as will be explained later) is evaluated, the result is a technical statement about the strength of the system. This statement is made (usually) without regard to the specific environment in

¹By "reference monitor concept" we mean strictly the concept of an abstract machine that mediates all accesses of subjects to objects. We do not mean to imply "reference validation mechanism", "security kernel", or even "Bell and LaPadula model".

²For the purposes of this paper, an AIS is any system which is used to create, prepare, or manipulate information in electronic form.

which the system will be operated, and all systems with the same rating meet the same criteria. No such statement can be made about an "interconnected accredited AIS"; all that can be provided is technical guidance to an accreditor about certain rules to follow in hooking up components. The technical statement provided by an evaluation is much stronger than any interconnection rule, and leads to much more confidence that the system will behave properly when it is installed.

Why Is It an "Interpretation" ?

It is a simple statement of fact that the TCSEC actually contains two things. First, it contains the general requirements for a trusted system OF ANY TYPE. Second, it contains an interpretation of those requirements for general-purpose operating systems. In some ways, it is unfortunate that these two things are so tightly interweaved throughout the document, but that is the way the document was written. Since the TNI is an interpretation of the general requirements for networks, it is on the same level as the interpretation for general-purpose operating systems in the TCSEC. That is, it is much more than a "Guideline". However, the TNI is an "Interpretation" rather than a "Criteria" because it interprets the general requirements, which have already been stated by the TCSEC.

Structure of the Document

The TNI is divided into two parts, plus three appendices. Part I of the document contains the TCSEC interpretations. For each requirement in each class, the requirement is stated as it appears in DoD5200.28-STD. Then, the interpretation of the requirements is stated. Finally, rationale is provided--an explanation of why the interpretation is as it is. For some requirements, examples of acceptable mechanisms are also provided.

Part II contains the requirements for security services such as Communications Field Integrity, Non-Repudiation, Continuity of Operations, and Network Management. Part II includes discussions of general assurance

factors, documentation requirements, and how to determine which services are needed in a particular application.

Appendix A discusses the evaluation of components. Appendix B provides the technical rationale behind the partitioned NTCB approach. Appendix C discusses considerations involved in the Interconnected Accredited AIS view. There is also a list of acronyms used in the document, and a glossary of terms.

Relationship to ISO Work

An effort is underway to extend the ISO Open System Interconnection (OSI) architecture by defining security-related architectural elements which can be applied in the circumstances for which protection of communications is required [5]. There is considerable overlap between the OSI Security Addendum and Part II of the TNI. Since at the time of this writing both documents are evolving, it is difficult to exactly define the relationship. However, some of the security services identified in the ISO addendum are addressed in Part I of the TNI, while others are addressed in Part II. The principle difference is that the ISO work is primarily concerned with Functionality, somewhat concerned with Strength of Mechanism, and rarely concerned with Assurance. The TNI, like the TCSEC before it, is very concerned with Assurance.

Part I: The TCSEC Interpretations

As its name suggests, Part I of the TNI consists of the interpretations of the TCSEC requirements. The working group has gone through the TCSEC, class by class and requirement by requirement, and asked, "What does this mean when the context is a network, rather than a general-purpose operating system?" Part I first restates the requirement, as it appears in DoD 5200.28-STD. The interpretation of the requirement is then stated, followed by the Rationale for the Interpretation. In certain cases, the Rationale also includes examples of mechanisms that may be used to satisfy the requirement. These examples are meant to be just that; they

are not meant to be prescriptive.

This interpretation makes explicit what is implicit in the TCSEC: that the Criteria can be applied to mandatory and discretionary integrity policies, just as it can to mandatory and discretionary secrecy policies. That is, it is permissible for a network system to support a secrecy policy, an integrity policy, or both.

The evaluation system for Part I of the TNI is identical to that for the TCSEC. A single, digraph rating in the range D to A1 is assigned to the system. This rating is a technical statement of the amount of trust that can be placed in the network system. It carries the same meaning as the digraph rating assigned to a general-purpose operating system that has been evaluated against the the TCSEC.

Part II: Other Security Services

Why Other Security Services?

Part II contains additional network security concerns that are not reflected in Part I. These concerns are what differentiate the network environment from the stand-alone computer environment. Some concerns take on increased significance in the network environment; others do not exist in stand-alone computers. Some of these concerns are outside the scope of Part I; others lack the theoretical basis and formal analysis underlying Part I. Since introducing these services into Part I would destroy the cohesiveness of the criteria for a class, they are treated separately in Part II.

Criteria Form: Functionality, Strength, and Assurance

Functionality refers to the objective and approach of a security service; it includes features, mechanisms, and performance. Strength of mechanism refers to how well a specific approach may be expected to achieve its objectives. Assurance refers to a basis for believing that the functionality will be achieved; it includes tamper resistance, correctness, verifiability, and resistance against circumvention or bypass.

As an example, consider communications integrity protection against message modification. A functionality decision is to select error detection only or detection and correction. A strength of mechanism decision would involve how strong an algorithm to use in implementing whichever were chosen. Assurance decisions would involve what level of software engineering would be involved in building the services, whether or not to use formal verification, and what level of testing to use.

For each of the security services described in Part II, requirements are given for each of Functionality, Strength of Mechanism, and Assurance. These requirements are distinct from one another, and may be met independently. For example, it may be decided to implement a very strong mechanism with very low assurance, or a very weak mechanism with very high assurance.

The Evaluation System

The security services described in Part II are not as strongly intertwined as are those in Part I. It is not possible to assign one rating (e.g., 'Z1') that adequately reflects how well the system provides each service. Furthermore, the services in Part II are generally not provided by the NTCB, but are provided by hardware/software that is external to the NTCB. To try to assign them a rating that is one of the digraphs assigned under Part I of the TNI is not practical, since in many cases the rating assigned is much more subjective. Therefore, a qualitative rating system must be used, instead of a hierarchically-ordered system. The evaluation system used in this document involves a tuple. A system is assigned three ratings for each service: one each for Functionality, Strength of Mechanism, and Assurance. Ratings normally come from the set of {Not Offered, None, Minimum, Fair, Good}; however, in specific cases, ratings such as "present" or "approved for use with data up to SECRET" may be assigned.

The difference between "Not Offered" and "None" is that a rating of None states that the system sponsor attempted to provide the

service (either Functionality, Strength of Mechanism, or Assurance) and failed completely. A rating of Not Offered merely implies that the sponsor did not attempt to provide the service, as (s)he did not consider it important. Since either rating indicates that a system does not adequately provide a service, the only appreciable difference to the potential customer is that a rating of None may indicate a poor quality of work in the system.

Selecting Security Services

Not all security services will be equally important in any specific environment; nor will their relative importance be the same among different environments. The system's accreditor (or the potential customer) must decide, based on the threats to be encountered in his/her specific environment, which security services are important, and which are not required. (S)He can then decide whether the rating achieved by a specific product is adequate for the projected environment.

General Assurance Approaches

There are a number of factors that involve the Assurance ratings of several security services. These assurance factors include such things as service design and implementation, service testing, design specification and verification, and configuration management. When a service is implemented, the rating for these general assurance factors is combined with the rating for the service-specific assurance factors to produce one overall Assurance rating for the service.

Supportive Primitives

There are two mechanisms/assurance techniques that apply across a wide range of services. These are encryption and protocols. Encryption is a tool for protecting data from compromise or modification attacks. The analysis of encryption algorithms and implementations is quite different from the analysis of most of the other requirements in

the TNI. The TNI states that assurance of encryption techniques will be provided by the National Security Agency.

Protocols are a set of rules and formats which determine the communication behavior between entities in a network. Many network security services are implemented with the help of protocols. Failure in the protocol therefore results in failure of the service. Protocols influence all ratings; there are Functionality factors, Strength of Mechanism factors, and Assurance factors involved.

General Documentation Requirements

Documentation is required for security services, just as it is for the NTCB. In fact, in many cases, the documentation should be contained in the same place. For example, guidance to the system or component administrator concerning security services should probably be placed in the Trusted Facility Manual. If a component supports users, guidance to those users should be placed in the Security Features User's Guide required by Part I. Documentation concerning the design and testing of a security service may be placed with the Test Documentation and Design Documentation required by Part I; if it is not located there, then it must be provided separately by the network sponsor.

Specific Security Services

The three categories of security services addressed are Communications Integrity, Denial of Service, and Transmission Security. Communications Integrity is further broken down into: Authentication, Communications Field Integrity, and Non-repudiation. Denial of Service contains the requirements for Continuity of Operations, Protocol-based Protection, and Network Management. Transmission Security includes Data Confidentiality, Traffic Confidentiality, and Selective Routing.

In Part II, Authentication is concerned with what the ISO work calls Peer Entity Authentication or Data Origin Authentication, depending on whether the service is

connection-oriented or connectionless. This can be contrasted with the Authentication required in Part I, which is strictly the Identification and Authentication of human users.

Communications Field Integrity refers to the protection from modification of any or all fields involved in communications. Non-repudiation provides unforgeable and undeniable proof of shipment and/or receipt of data.

It is accepted that one can never completely protect against denial of service. Furthermore, the TNI does not attempt to address protection against such attacks as cutting a communications cable, or blowing up one of the components. The TNI does state requirements for detecting service levels that have fallen below pre-established thresholds, and for detecting the fact that access to a component is unavailable.

Transmission security is a collective term for a number of security services. These services are all concerned with the secrecy of information transfer between peer entities through the computer communications network. While physical security can also provide transmission security, it is not explicitly addressed in the TNI.

Appendix A: Component Evaluations

The main body of the TNI takes the view of a network as a single trusted system. This view can be extended somewhat, and a trusted network can be regarded as a collection of trusted components. This is an important extension, as in the commercial marketplace it is doubtful that many vendors will provide complete systems. Thus, we would like to be able to assess the trust provided by different types of components. There are two advantages to being able to do this: first, it allows for the evaluation of components which in and of themselves do not support all of the policies required by the TCSEC; second, it allows for the reuse of the evaluated component in different networks without the need for a re-evaluation of the component.

Taxonomy of Policies and Components

For our purposes, there are four basic types of policies that systems or components can enforce. There are mandatory access control policies, discretionary access control policies, supportive policies, and application policies.

Application policies are those that apply to specific programs; they provide security in addition to that provided by the TCB or NTCB partition. An example of an application policy would be a database management system that provided access control to the record or field level, while the TCB provides access control only to the granularity of a file. Application policies are not relevant to the TNI; thus they will not be addressed.

Supportive policies include identification and authentication policies as well as audit policies.

Given this taxonomy of policies, the TNI breaks the universe of components into four classes. One class consists of those components that support mandatory access control policies; the TNI denotes these 'M components'. A second class consists of those components that support discretionary access control policies; the TNI calls these 'D components'. The third class supports identification and authentication policies, and these are called 'I components'. The final class supports audit policies; these are called 'A components'.

Evaluation System

Whenever a component is to be evaluated, the component sponsor is responsible for completely defining a target network architecture; that is, an architecture in which the component is expected to be used and for which its security features will work as stated. Once this is done, the component can be evaluated against those requirements that apply to it, in the context of the stated target architecture and policy.

A component is evaluated against the

requirements in Part II as stated for any service it provides. No further interpretation is necessary.

A component is evaluated against some subset of the requirements for a given class in Part I. It is evaluated against all assurance requirements, plus those feature requirements that apply directly to the policy it enforces. In general, the component is evaluated against the Interpretation as stated in Part I of the TNI. In some cases, it is necessary to reinterpret the requirement to place it in the context of a network component, rather than a network system.

The range of ratings that can be assigned to a component depends on the policy(ies) it enforces. For example, a M component can receive a rating in the range B1 - A1. A D component can be rated from C1 to C2+. (C2+ indicates that the component enforces the B3 DAC requirement, and provides C2 assurances. It is not correct to assign a B3 rating to a D component, as that connotes a level of assurance that no D component can provide.) An A component can receive a rating of C2 or C2+, and an I component can be rated C1 through C2+.

Composition Rules

Since a component is defined to be any part of the system, some components are made by composing other components. For example, a communications subnet is a component of a larger system; it may be composed of packet switches, front-end units, and gateways that are components themselves. (This is an illustration of the fact that the definition of component is a recursive one.) In general it is not possible to guarantee that a collection of evaluated components will result in an evaluable trusted system. However, it is possible to define a set of composition rules so that the result of composing trusted components maintains the ratings assigned to the original components.

An example of the composition rules provided in the TNI is illustrated as follows. Suppose that there is a D component that has been given a C2 rating for D. Suppose that there

is an I component that has been given a C2 rating for I. We wish to compose these two components to get one DI component that is rated C2 for D and C2 for I. In order to do that, we must insure that the DI component preserves the Network DAC Policy of the D component. Furthermore, the DI component must preserve the Audit interface(s) used for exporting audit information from both the D component and the I component. If the DI component provides Identification/Authentication support services to other components, the Identification Interface of the DI component must be defined and a protocol established for this interface which is able to support the Network I/A Policy. If the DI component does not provide Identification/Authentication support services to other components, it may only be composed with other components which are self-sufficient with respect to DAC.

The TNI gives composition rules for interconnecting all possible combinations of component types, most of which are similar to the one above.

Appendix B: Rationale for the Partitioned NTCB Approach

Implicit in the partitioned NTCB approach is the view that a network, including the interconnected hosts, is analogous to a single trusted system, and can thus be evaluated using an interpretation of the TCSEC. Put another way, networks form an important and recognizable subclass of ADP systems with distinctive technical characteristics which allow tailored interpretations of the Criteria to be formulated for them. Appendix B provides the background and rationale for the partitioned NTCB approach.

Appendix C: The "Interconnected Accredited AIS" View

The interconnected accredited Automated Information System (AIS) view is an operational perspective that recognizes that parts of the network may be independently created, managed, and accredited. Each AIS is accredited to handle sensitive information at

a single level or over a range of levels. In this view, the individual AIS may be thought of as "devices" with which neighboring components can send and receive information.

The interconnected accredited AIS view differs from the single trusted system view in that, here, one does not regard a network as a single trusted system, and therefore one does not assign a single rating to the network. An example of where the interconnected accredited AIS view is necessary is a network consisting of two A1 systems and two B2 systems, all of which are interconnected and all of which may be accessed locally by some users. It is easy to see that, if we regard this as a single trusted system, it would be impossible for it to achieve a rating against Part I of this document higher than B2. This might not be an accurate reflection of the trust that could be placed in the two A1 systems and interconnections between them. Any single rating assigned to this network would be misleading.

Component Connections and the Interconnection Rule

Networks like the one described above can only be addressed in terms of whether or not they obey an interconnection rule. Each component that is connected to other AIS communicates by means of a particular I/O device, which has a device range associated with it. The interconnection rule involved is one that says simply, for two way communication, the device ranges of the two I/O devices must be identical. For one-way communication (i.e., with no acknowledgement whatsoever), the device range of the receiving I/O device must dominate the device range of the sending I/O device.

This interconnection rule must be enforced locally by each component of the network. Decisions on whether to send or receive information can be made by a component based only on its accreditation range and those of its immediate neighbors. In many cases, it is not necessary for a sending component to know the accreditation range of the component that is the ultimate destination of the message. If the interconnection rule is enforced by

each component, the overall network will prevent information from being sent where it shouldn't go.

The Global Network View

In many cases, networks can enforce the interconnection rule and still expose information to an excessive risk of disclosure or modification. There are considerations other than the interconnection rule that the accreditor may wish to take into account when deciding whether or not to permit interconnection of components. Most of these considerations are based on a knowledge of all the components in the network. As one particular example of these considerations, let us consider something called the "cascading problem". Cascading occurs when a penetrator can take advantage of network connections to compromise information across a range of security levels that is greater than the accreditation range of any of the component systems he must defeat to do so.

Consider the following example: there are two class B2 systems, one (System A) processing SECRET and TOP SECRET information, the other (System B) processing CONFIDENTIAL and SECRET information. A penetrator is assumed to be able to overcome the protection mechanisms in System A, causing TOP SECRET information to be downgraded to SECRET; have it sent across to System B at the SECRET level; and then overcome the protection mechanisms in System B to downgrade it to the CONFIDENTIAL level. TOP SECRET information has thus been downgraded to the CONFIDENTIAL level. According to the environments guidelines [6], the risk of this requires at least a class B3 system; however, the penetrator has only had to defeat two class B2 systems.

The TNI describes two heuristic algorithms for determining the presence of cascading conditions. One, which is very simple, is fairly conservative, and sometimes indicates the presence of a cascading condition when in fact none exists. The second is much more complex, but it tends to be more accurate in determining cascading conditions.

There are several ways of remedying potential

cascading conditions. In most cases, using a higher level of trusted system will suffice. In other situations, mechanisms such as end-to-end encryption will solve the problem. In extreme cases, the accreditor may wish to actually disallow the connection.

System Evaluation Criteria in Specific Environments, CSC-STD-003-85, 25 June 1985.

For More Information:

The author can be contacted at the following address:

Acknowledgements

The author would like to thank the members of the TNI working group, without whose efforts the TNI would not have been possible. They are: Marshall Abrams and Jon Millen, of MITRE; Roger Schell, of Gemini Computers, Inc.; Stephen Walker, of Trusted Information Systems, Inc.; Robert Morris, Chief Scientist of the NCSC; Irv Chatlin, NCSC; Jack Moskowitz, NCSC; and Brian Snow, Department of Defense. Acknowledgement is also due to Leonard LaPadula, Bill Shockley, Steve Padilla, and Jim Anderson, for their many and valuable inputs; and to Patrick Mallett, Albert Jeng, and Sam Schaen for review and comments. Thanks are also due to all of those who read and commented on various drafts of this document.

Alfred Arsenault
National Computer Security Center
ATTN: C11
9800 Savage Road
Ft. George G. Meade, MD 20755-6000

References

1. DoD Computer Security Center, Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security, New Orleans, LA 19-22 March 1985.
2. DoD Computer Security Center, Draft Trusted Network Evaluation Criteria, 25 July 1985.
3. National Computer Security Center, Draft Trusted Network Interpretation, 8 April 1987.
4. DoD Standard 5200.28-STD, Department of Defense Trusted Computer System Evaluation Criteria, December 1985.
5. "ISO 7498/Part 2 - Security Architecture," ISO/TC97/SC21/N1528/WG1 Ad hoc group on Security, Project 97.21.18, September 1986.
6. DoD Computer Security Center, Computer Security Requirements--Guidance for Applying the Department of Defense Trusted Computer

CONSIDERATIONS FOR SECURITY
IN THE OSI ARCHITECTURE

Dennis K. Branstad

Institute for Computer Sciences
and Technology
National Bureau of Standards
Gaithersburg, Maryland, 20899, USA

I. Introduction to OSI Security

The Open Systems Interconnection (OSI) computer network architecture has given computer network designers and implementors a common vocabulary and structure for building future networks. It has also given network security designers a foundation upon which desired security services can be defined and built. This paper discusses several goals of security in the OSI architecture as well as where and how the security services that satisfy them could be implemented.

A. Need for a Security Architecture

A standard security architecture is needed in OSI in order to begin the task of implementing security services in commercial products so that not only can one OSI system communicate with another, but also it can do the communication with the desired security. The security goals and services discussed in this paper are predicated on the assumptions that sensitive or valuable data are being transmitted between systems in the OSI network, that changes in the network between the systems could be made by an unauthorized person or persons in order to obtain or modify the data, and that security services are to be available in the network to prevent the unauthorized disclosure of sensitive data and to detect (and report) the unauthorized modification of data.

For this paper, security is defined to be the protection of the confidentiality and integrity of data. Privacy, often combined with security or confused with security, is a social issue regarding protection of personal information from undesirable use and is not discussed in this paper. Security is often defined as including protecting the availability of data but is not included in the scope of this paper.

B. Requirements for Security

A large number of potentially desirable security goals in computer networks have been identified in the literature. The OSI Implementors Workshop Special Interest Group in Security (OSI SIG-SEC) is establishing a desirable set of security goals for implementors of OSI and the resulting list of desirable services to implement. This SIG is sponsored by the U. S. National Bureau of Standards and is open to anyone interested in OSI security.

NOTE: CONTRIBUTION OF THE NATIONAL BUREAU OF STANDARDS.
NOT SUBJECT TO COPYRIGHT.

A minimum set of desirable security goals in OSI identified by the author is:

1. Protection of data against unauthorized modification.
2. Protection of data against undetected loss/repetition.
3. Protection of data against unauthorized disclosure.
4. Assurance of the correct identity of the sender of data.
5. Assurance of the correct receiver of the data.

As a memory aid for these five basic security goals, the following five terms starting with the letter "S" have been selected to represent the security achieved by satisfying these goals. They are, respectively:

1. Sealed
2. Sequenced
3. Secret
4. Signed
5. Stamped

Achieving these security goals in the OSI architecture will assure that data being transmitted from one OSI system to another will not have been modified, disclosed, replayed, or lost in the network without the sender and/or the intended receiver being notified and that the participating parties in the communication have been correctly identified.

Other security goals that have been identified [11] as being desirable include: labeling of data according to its sensitivity, source, etc.; not disclosing the identities of the sender and recipient of data, and the quantity of data exchanged, except to each other; providing security audit trails of network communications; assuring the availability of communications under adverse conditions; assuring that data inside an OSI system cannot be transmitted using covert information channels, even of very low bandwidth; proving to an independent third party that a communication did occur and the correct contents were received; obtaining explicit authorization for access to a system before making a connection to the system.

C. National Bureau of Standard's Role

The National Bureau of Standards (NBS) has fostered the development of the OSI architecture and the implementation of commercial products implementing the standard protocols defined for the architecture. NBS has had a program in

computer security since 1973 and has fostered the development of numerous security standards [7, 8, 9, 10] since that time. It has assisted in the development of several security standards in the banking community [4, 5, 6] and the information processing community [1, 2, 3] through the American National Standards Institute. It is now supporting the development of an OSI security architecture [11] via the ISO/ TC97/ SC21/ WG1 and the OSI SIG-SEC.

II. OSI Network Security Perimeters

A useful notion in the development, implementation and use of security in a computer network is that of a security perimeter. This logical structure in a computer network is the equivalent to a physical structure in a secure facility such as a bank vault. In actuality there are multiple security perimeters around highly secure facilities where a principal of "security in depth" is practiced. Similar analogies can be drawn in computer networks. For simplicity in this discussion, a single security perimeter concept will be used in which each OSI system will have a security perimeter. The overall goal of OSI security is to communicate data from within one security perimeter to another. Loss of security within a perimeter is beyond the scope of this paper.

A. One Security Perimeter around Network

If a security perimeter is drawn around the entire network (Figure 1), either because no sensitive or valuable data are ever communicated in the network, because no threats are believed to exist in the network, or because security is provided through non-OSI methods, then no OSI security services are needed. Many networks are presently being operated in this manner. This is acceptable as long as everyone and everything inside the perimeter is "trusted." Trust implies that no intentional or accidental event will occur which will result in an undesirable disclosure, modification or loss of data. A simplified definition of trust is used in this paper with trust being a binary valued parameter (i.e., multi-level security is not considered). Trust can also be assured within the system through the use of a "Trusted Operating System." This system assures that adequate security is provided within the security perimeter.

P	User Processes	P
7	Application Layer	7
6	Presentation Layer	6
5	Session Layer	5
4	Transport Layer	4
3	Network Layer	3
2	Link Layer	2
1	Physical Layer	1

Figure 1: One Security Perimeter around network

B. Security Perimeter around each User Process

A security perimeter could be drawn around each user process which provides high granularity security (Figure 2) since each user process provides its own protection and nothing within the OSI architecture needs to be trusted. However, this requires that all desired security services be implemented in every user process or program. While possible, this approach is contrary to the goal of OSI for performing services in the layers of OSI rather than in each user process.

P	P
7	7
6	6
5	5
4	4
3	3
2	2
1	1

Figure 2: Security Perimeter around each User Process

C. Security Perimeter around Upper Layers

A security perimeter can be drawn between these two extremes around the upper layers of the OSI architecture. Different granularities of security result from selecting different placement of the security perimeter. In actuality, a hierarchy of security perimeters will be implemented, each providing security against a different perceived threat. A security perimeter has been drawn at the transport layer (layer 4) of the OSI architecture (Figure 3) for subsequent discussion in this paper

P	P
7	7
6	6
5	5
4	4
3	3
2	2
1	1

Figure 3: Security Perimeter around Upper Layers

D. Negotiated Security

One goal of OSI implementors should be to provide maximum flexibility for users of an implementation. An implementation should provide for negotiation between users in selecting an optimum set of OSI services, including security services. However, security may be somewhat unique in this regard in that some organizations may not desire to negotiate certain security services, especially if the negotiation could result in security less than some predetermined minimum. Other organizations may accept negotiating away all security services if those services are temporarily causing functionality or throughput to drop

below a minimum. Some organizations may add to the basic security services provided in standard implementations and not desire other organizations to use or know about the additional services.

An extensible security architecture is desired which will provide for these special services without causing an unacceptable overhead on those not requiring these services.

III. Placement of Security Services in the OSI Architecture

A. Security Addendum to the OSI Architecture

A draft security addendum to the OSI architecture [11] has been developed by Ad Hoc groups of the American National Standards Institute (ANSI) and the International Standards Organization (ISO) TC97/ SC21/ WG1. The draft security addendum presents a glossary of computer security terms, describes a number of security services for OSI, and presents a matrix of where in the seven layer OSI architecture the security services may be located (See Below). It then presents the rationale for why the security services are placed in those layers. Recent work [12] defines an authentication framework for the layer 7 directory service for which User Agents are authenticated before they are granted access to sensitive information in the Directory.

While the draft addendum satisfies the goals of defining a number of security services and discussing where they could be placed, the addendum is not adequate for an implementor desiring to implement security in the OSI architecture. First, it would be too expensive to provide all security services at all possible layers allowed in the addendum. Second, if one implementor chose to implement a service at one layer and another implementor chose to implement the same service at a different layer, the goal of compatibility between peer layers of OSI would not be achieved. Finally, standards for implementing the services are not currently specified.

B. OSI Security Categories and Services

The following security categories and services are defined in the draft security addendum to the OSI architecture. The OSI layers in which the services could be implemented are shown in the matrix next to the services. The services need not be implemented in all of the layers that are specified.

OSI SECURITY SERVICE PLACEMENT PRIORITIES

High (H); Medium (M); Low (L)

OSI LAYER							CATEGORY OF SERVICE
1	2	3	4	5	6	7	SERVICE

1. IDENTIFICATION/AUTHENTICATION

_ _ M L _ _ H	A. Data Origin
---------------	----------------

_ _ L M _ _ H	B. Peer Entity
---------------	----------------

2. ACCESS CONTROL

_ _ M L _ _ H	A. Originator Authorization
---------------	-----------------------------

_ _ L M _ _ H	B. Peer Entity Authorization
---------------	------------------------------

3. INTEGRITY

_ _ _ H _ _ _	A. Connection (w/wo error recovery)
---------------	-------------------------------------

_ _ H M _ _ _	B. Connectionless (wo error recovery)
---------------	---------------------------------------

_ _ _ _ _ H	C. Selective Field Integrity
-------------	------------------------------

4. CONFIDENTIALITY

_ _ _ H _ _ _	A. Connection
---------------	---------------

_ _ H M _ _ _	B. Connectionless
---------------	-------------------

_ _ _ _ _ H _	C. Selective Field
---------------	--------------------

H _ _ _ _ _ _	D. Traffic Flow
---------------	-----------------

5. NON-REPUDIATION

_ _ _ _ _ _ H	A. Originator
---------------	---------------

_ _ _ _ _ _ H	B. Recipient
---------------	--------------

C. Factors in Placing Security Services

Many factors must be considered in selecting the layer(s) for implementing selected security services. First, a basic set of security services to be implemented must be chosen. Second, a minimum number of layers should be chosen in which to implement the services to minimize the number of layers affected by security. Third, use of existing services of a layer may be utilized by the security service if a proper layer is chosen. Fourth, the overall cost of providing the selected security services will be minimized if the layer is properly selected. Fifth, a set of primitive security functions need to be defined and implemented (hardware, software, firmware) in such a way that they can be performed at one or more layers of the architecture in providing the desired security service.

D. Primitive Security Functions

OSI security services could be implemented utilizing a set of primitive functions similar to the ones below. The primitive functions would be called with a set of parameters enclosed in [] and return the results enclosed in { } following execution.

I. AUTHENTICATE [ID; AUTHENTICATOR]
{RESULT; STATUS}

This primitive verifies that the AUTHENTICATOR does correspond with the claimed ID by searching the local Secure Management Information Base and responding with the correct RESULT and STATUS.

II. AUTHORIZE [ID; TYPE; RESOURCE]
{RESULT; STATUS}

This primitive verifies the authorization of ID with the indicated TYPE for access to the requested RESOURCE and sets the correct RESULT and STATUS.

III. ENCRYPT [PT; LENGTH; KEYNAME] {CT; LENGTH; STATUS}

This primitive enciphers plaintext beginning at PT for the indicated LENGTH into ciphertext beginning at CT for the indicated LENGTH and sets the resulting STATUS using the KEY associated with KEYNAME.

IV. DECRYPT [CT; LENGTH; KEYNAME] {PT; LENGTH; STATUS}

This primitive decipheres ciphertext beginning at CT for the indicated LENGTH into plaintext beginning at PT for the indicated LENGTH and sets the resulting STATUS using the KEY associated with KEYNAME.

V. COMPUTEMAC [DATA; LENGTH; KEYNAME]
{MAC; STATUS}

This primitive computes a Message Authentication Code (MAC) on the DATA of indicated LENGTH using the KEY associated with KEYNAME and sets the resulting STATUS.

VI. VERIFYMAC [DATA; LENGTH; KEYNAME; MAC] {RESULT}

This primitive computes a Test Message Authentication Code (TMAC) on the DATA of indicated LENGTH using the KEY associated with KEYNAME and sets the correct RESULT to indicate if TMAC is identical with the input MAC.

VII. SIGN [DATA; LENGTH; USERID; KEYNAME] {SIGNATURE; STATUS}

This primitive computes a SIGNATURE on the DATA of indicated LENGTH for the user indicated by USERID using the KEY associated with KEYNAME and sets the resulting STATUS.

VIII. VERIFYSIGNATURE [DATA; LENGTH; USERID; KEYNAME] {SIGNATURE; RESULT; STATUS}

This primitive computes a Test Signature (TSIGNATURE) on the DATA of indicated LENGTH for the user indicated by USERID using the KEY associated with KEYNAME, compares it with SIGNATURE, and sets the correct RESULT and STATUS.

E. Initial Recommendations for Placement

Based on the simplifying assumptions stated at the beginning of this paper, the transport layer (4) of the OSI architecture was chosen by NBS for initial implementation of a selected subset of security services. This layer was chosen after several years of participating in the development of standards for security at layers 1/2 [2], layer 4 [13] and layer 6 of the OSI architecture by the accredited ANSI Technical Committee X3T1. The layer 1/2 standard was developed for protecting data in each link of a network. However, it does not provide security from one OSI end-system computer to another through a general network. A layer 4 standard was drafted to provide security for all data in a layer 4, class 4 connection. A layer 6 standard was drafted to provide security for selected fields of data specified by an application in such a way that it need not be unprotected even at the intended destination.

Early development of the layer 4 standard was facilitated by an early definition of services at layer 4 and the existence of standard protocols and implementations of layer 4. It was also facilitated by using existing services of layer 4 for security purposes.

IV. Protocols for Transport Layer Security Services

A. Integrity Service

A connection integrity service protocol has been defined for class 4 of the transport layer (4) of the OSI architecture. The integrity service can achieve two security goals, sealing and sequencing, and assures that all data in a connection are transferred from one OSI security perimeter to another without being intentionally or accidentally modified, lost or repeated. Such security is especially important in Electronic Funds Transfer (EFT) transactions. EFT messages are vulnerable to modification; deposit and withdrawal messages are vulnerable to loss or repetition. While present EFT security standards specify security services at layer 7 of the OSI architecture, a wide variety of other applications could utilize similar security services if they are implemented at layer 4.

The integrity service protocol utilizes the sequence number provided by layer 4, class 4 service. This is a 31-bit number defined as 4 octets in the header of each layer 4 Protocol Data Unit (PDU). The sequence number is provided by layer 4 for resequencing the PDUs if they arrive out of order and for flow control on a connection. The integrity service also utilizes the existing layer 4, class 4 mechanisms for recovery from errors (i.e., lost or modified data). Connectionless network layer (3) services can then be used if a class 4 integrity service is provided and used at layer 4.

The PDU integrity protocol specifies how an electronic data integrity seal, called a Message Authentication Code (MAC), is computed for each PDU. The seal covers both the user data and the header (including sequence numbers) for data stream integrity. The seal is typically a 32-bit number that is computed using cryptographic functions on the PDU to be sealed so that its integrity can be verified when it is received at the corresponding security perimeter (layer 4 peer entity). If any part of the PDU has been accidentally or intentionally modified, including the address and sequence number, the test value computed on the received PDU will not match with the seal computed by the transmitter on the transmitted PDU and transmitted with the sealed PDU. If the value is not correct, the suspected PDU is discarded and a retransmission is requested. If the value is correct, the PDU is accepted. Sequence numbers are also verified to assure data stream integrity.

B. Confidentiality Service

Data can be protected against unauthorized disclosure in a network with encipherment (encryption). The ISO/OSI security addendum calls this a confidentiality service. Enciphering is a transformation of data into a form that is not usable or readable while preserving the information content. The resulting ciphertext is transmitted. The authorized receiver must perform the correct inverse operation, called deciphering (decryption), in order to obtain the original, usable, readable form of the data. Typically, a cryptographic algorithm, implemented in a computer with either hardware, software or both, and a cryptographic variable called a key are used to perform the two required transformations. A requirement of this service is that something be kept secret or available only to authorized communicating parties. Details of this service are beyond the scope of this paper.

The confidentiality service requires that the user data of a PDU be enciphered before leaving the security perimeter of the transmitter and be deciphered only after entering the security perimeter of the intended receiver. Other portions of the PDU need not be enciphered since they contain no user data. If enciphering is performed only on the user data, the addresses or identities of the communicating parties are not enciphered and hence a monitor in the network can determine who is communicating and how much data is being communicated, even though the contents of the data cannot be determined.

The OSI security architecture specifies a traffic flow confidentiality service at layer 1 to protect against traffic analysis if this protection is desired. Encipherment at this layer would protect all data on a communication link, including the addresses of the communicating entities. However, it would be unprotected in all intervening gateways.

C. Peer Authentication Service

The two communicating transport layers are called peer entities and must perform equivalent services in order to communicate. Simplistically, what one does the other must check and/or undo. The security protocols that have been defined to date at layer 4 will assure that the peer layers are mutually identified and that a connection between them is a current connection and not a replay of a previous connection. This protocol relies on cryptographic procedures during the establishment of a connection. Once a connection is established, data intended for the peer layer 4 can only be used by that peer entity. It can be accidentally or intentionally destroyed, delayed or misrouted, but it cannot be used by the unauthorized receiver if encrypted.

Peer authentication is performed by a connection procedure often called a three-way handshake. Using proper cryptographic procedures, a challenge-response-verification is performed by both peer entities of a connection. Random numbers are used in a standard procedure to assure that both peer entities have the correct key and that a replay of a previous connection is not being attempted. The user data is not signed with this technique. The personal identities of the users of a connection or the applications using a connection are not involved in this service. It merely assures that an entire stream of data is not replayed to an unsuspecting recipient.

V. NBS Laboratory Implementation

A. Local Area Network Environment

The National Bureau of Standards initiated an experiment in implementing these security protocols in the transport layer of several computer systems in a local area network environment. The experiment was to determine the adequacy of a proposed ANSI standard for the security protocols, the ease of implementation and impact on the operation of the network.

The network was based on one of the IEEE 802 standards often called Ethernet. Six personal computers were used for the experiment. Ethernet circuit boards were added to the computers and connected together using coaxial cable. Software supplied with each Ethernet board was used to provide layer 1, 2 and 3 functionality. A transport layer protocol that was implemented on a time-shared mini-computer was used as the basis of the experiment. Null layers 5 and 6 were used. A simple layer 7 application was used to demonstrate connections and data transfers among the computers.

The National Bureau of Standards Data Encryption Standard (DES) was used for the cryptographic functions. Six circuit boards each containing DES devices were obtained from two companies and plugged into the six personal computers. These boards were used by the layer 4 security services. Cryptographic keys for each of the six computers were manually installed in the computers for demonstrations. No automated key management was performed during the experiment.

B. Lessons Learned

The difficulty of converting a protocol designed for a time-shared, interrupt driven mini-computer to a single-user, event driven personal computer was not anticipated. Even though the programming language was the same on both systems, it was found to be very difficult to convert the program from one system to another. A completely new system interface had to be developed in order to use the services of the transport protocol.

It was found to be easy to integrate the security services into the transport protocol once the protocol was working. The confidentiality service was the easiest to implement. The integrity service was the most difficult as it required more modifications of existing layer 4 functions. The peer authentication service was trivial after implementing the integrity service. Since the system was designed only for demonstration, there was no attempt to verify the correctness and trust of the implementing code itself which would be necessary for operational systems.

It was difficult to effectively demonstrate security of the network. Good security implementations should have minimal effects on the user and the network. It was often impossible to tell if the security services were being performed since they caused negligible overhead on the network. A network monitor was finally designed to observe the data on the network so that security services, or lack thereof, could be observed.

It was acceptable to have special applications to demonstrate the security services and the transport services but it was apparent that original equipment and software implementors and vendors have to support the enhanced security functions as a basic feature of their product in future products in order to gain the desired security and user support. The interface to security enhancements has to be trusted and integrated into the product or security will often be bypassed.

VI. Summary and Conclusions

A security architecture is needed as a fundamental part of the OSI architecture. Standard security services must be defined, standard security protocols must be developed and standard security interfaces for applications programs must be specified. Optional security services must be defined and standard implementations must be available to be used on an optional basis. All security services need to be negotiated but with provisions for default services and enhanced, user defined services. The user should not be aware of the operation of security services other than the need for providing initial information for the service (e.g., the set of services required, specific parameters for the service if default parameters are not acceptable).

While only a small subset of the possible desirable security services were selected for discussion in this paper, there is a need for research in providing additional services and for standards activities for specifying implementations of them. The National Bureau of Standards is seeking interest and assistance in providing these necessary activities.

VII. References

- [1] ANSI X3.92, American National Standard for Information Systems - Data Encryption Algorithm, American National Standards Institute, New York, NY, 1981.
- [2] ANSI X3.105, American National Standard for Information Systems - Data Link Encryption, American National Standards Institute, New York, NY, 1983.
- [3] ANSI X3.106, American National Standard for Information Systems - Data Encryption Algorithm Modes of Operation, American National Standards Institute, New York, NY, 1983.
- [4] ANSI X9.8, American National Standard for PIN Management and Security, American National Standards Institute, New York, NY, 1982.
- [5] ANSI X9.9, American National Standard for Financial Institution Message Authentication - Wholesale, American National Standards Institute, New York, NY, 1986.
- [6] ANSI X9.17, American National Standard for Financial Institution Key Management - Wholesale, American National Standards Institute, New York, NY, 1985.
- [7] Federal Information Processing Standard 46: Data Encryption Standard (DES), National Bureau of Standards, Gaithersburg, MD, 1977.
- [8] Federal Information Processing Standard 74: Guidelines for Implementing and Using the Data Encryption Standard, National Bureau of Standards, Gaithersburg, MD, 1980.
- [9] Federal Information Processing Standard 81: DES Modes of Operation, National Bureau of Standards, Gaithersburg, MD, 1980.
- [10] Federal Information Processing Standard 113: Computer Data Authentication, National Bureau of Standards, Gaithersburg, MD, 1985.
- [11] ISO 7498: Proposed Draft Addendum Number 2 - Security Architecture, ISO/TC97/SC21/WG1, 1986.
- [12] The Directory - Authentication Framework, ISO/CCITT Directory Convergence Document #3, ISO/TC97/SC21/WG4, 1986.
- [13] Transport Layer Protocol Definition for Providing Connection Oriented End-to-End Cryptographic Data Protection Using a 64-Bit Block Cipher, X3T1 Draft Document forwarded to ISO TC97/SC20/WG3, 1986.

A Mission-Critical Approach to Network Security

Howard L. Johnson
Information Intelligence Sciences, Inc.
15694 E. Chenango
Aurora, CO 80015

J. Daniel Layne
Computer Technology Associates, Inc.
7150 Campus Drive, Suite 100
Colorado Springs, CO 80918

ABSTRACT

Computer networks supporting command and control missions interconnect sensors, operations centers, forces and other heterogeneous systems. Such "systems of systems" must protect sensitive data from the threat of compromise and must, in addition, provide protection to mission critical data and resources against loss-of-integrity and denial-of-service. This paper presents an approach to network security that treats sensitivity (classified data protection) issues independent of criticality (integrity and availability) issues to gain architectural and economic advantage. Decomposition of large systems into components is reviewed. We discuss protection mechanisms to counter sensitivity and criticality threats and also address security interface policy requirements between components and systems. Finally, a network security architecture concept is suggested.

INTRODUCTION

Networks and more specifically distributed systems present a more difficult security problem than monolithic computer systems due to lack of central control and a heightened security exposure that is geographically dispersed and over a broader range of levels. Communicating components compound the problems with different security policies and interfaces, incompatible security architectures, and composite risks. There is a lack of strong technology history in network security and external exposure of communications media and facilities provides greater opportunity for integrity and denial of service attacks.

In communications systems we protect data content exposure with cryptography, but without additional protection (not presently provided in computer security), false messages can be initiated, important messages can be deleted, and communications resources could be made unavailable. To make matters more difficult, a probable profile of today's enemy is someone who has a security clearance, has dedicated many years service to the Government, and possesses detailed technical knowledge of computer hardware and software.

We examined DoD's derived security policy and found that it primarily addresses monolithic computer systems in a protected environment. It is not definitive where

complexity exists and deals principally with information protection issues (not mission protection issues). Further, connecting equipment in DoD installations appears to be leading to the requirement for all highly critical/highly classified systems to be certified/accredited at least to the A1 (Orange Book [1]) level. This is a technologically difficult goal that magnifies development cost and can impose in its solution unacceptable operational constraints and risks.

This paper separates sensitivity (protection of classified information) from criticality (integrity of operations and protection against denial of services). This decision results in the ability to use encryption and covert channel protection mechanisms to solve the sensitivity problem in host communications and data storage, leaving the criticality problem to be addressed. Criticality can generally be solved in networks with detection and recovery approaches, existing primarily in the host protected domain; which is far less costly than resistive (formal model) mechanisms. We believe this solution will not only reduce operational constraints, but will also provide a less expensive approach to even a higher level of security.

SENSITIVITY AND CRITICALITY

The separation of network security into sensitivity and criticality follows the partial lead of Air Force Regulation (AFR) 205-16 [2]. Figure 1 illustrates the key elements of sensitivity and criticality. Sensitivity is generally well understood while criticality has two defining aspects,

- o maintaining the integrity of the system to ensure that senders and receivers are as perceived, that processes (e.g., protection, communications, and resource control) are as intended, and that data (mission or control) have not been altered; and

This work was sponsored in part by the USAF Space Command, under contract number F05604-85-C-0019 awarded to CTA. However, the statements herein reflect the opinions of the authors, and do not necessarily represent the views or policy of the Air Force.

Criteria Topic	Sensitivity (Existing Basis)	Criticality (Proposed Enhancements)
Protect	Classified data	Mission Data Control Data, Processes
Threat	Disclosure	Loss of Integrity Denial of Service
Levels	Unclassified Confidential Secret Top Secret (Compartments)	Noncritical Critical Highly-Critical (Compartments possible)
Control Goal	Need-to-Know	Need-to-Modify/Execute
Protection Mechanisms	Resistance	Resistance Detection/Recovery

Figure 1. System Security Elements

- o protection against denial of service that occurs when unauthorized action prevents the system from providing normal and intended services for the mission.

Criticality and sensitivity are duals in at least one sense. A mission can be extremely critical and have no classified data (an important unclassified sensor), or a mission can deal with great amounts of classified data and not be nationally critical (such as a classified library).

In the history of formal computer security, integrity has only applied to the trusted computer base and denial of service was not seriously addressed. In networks, integrity and availability concepts apply to communication control (e.g., key distribution, protocols), control processes, and mechanisms (in a manner similar to the trusted computer base). Network requirements further pertain to distribution paths and options.

The entire control mechanism must be trusted to some degree. The concepts of detection and recovery apply to data and data path alteration, as well as to inappropriate or unauthorized resource use. The integrity of protection mechanisms, concerned with compromise of classified information, is a sensitivity issue, whereas the integrity of mechanisms that ensure authentication, trusted communications processes, and accuracy of control data is a criticality issue.

Criticality is defined in AFR 205-16 as "the required level of protection of resources, whose compromise, alteration, destruction, loss, or failure to meet objectives will jeopardize mission accomplishment." When we speak of criticality of a local mission, it must be taken in the context of DoD overall objectives.

Level of protection against compromise is commensurate with the information sensitivity (generally having global and long term mission implications). The value of protection against loss of integrity or denial of service threat is commensurate with

the mission that might not be accomplished and has current, short term, local implications (where local pertains, for example, to two parties attempting to communicate or to operations affected by resource unavailability).

The Trusted Base (system or component where a computer is usually a component) may consist of mechanisms for both sensitivity protection and criticality protection, however the degree of protection of the two might differ. In certain cases, the same mechanism can be used for both objectives while in other cases distinct mechanisms must be incorporated.

In criticality protection mechanisms, detection and recovery become more applicable than they are in sensitivity protection. In fact, these may be more important (and certainly more practical) than resistance. When a crisis occurs, there may be a tendency to treat sensitivity more lightly to enhance operational knowledge and flexibility (e.g., releasing data from normally protected intelligence sources for operational decisions). However, in general, during a crisis, criticality becomes increasingly important.

In AFR 205-16 the levels of criticality are dealt with more subjectively than the levels of sensitivity. If we follow the lead of Biba [4], specific graduated levels are identified and dealt with in the system with security mechanisms to protect the levels according to a set of rules, similar to the mandatory and discretionary policy in sensitivity protection. The levels identified in AFR 205-16 are: HIGHLY-CRITICAL, CRITICAL, and NONCRITICAL. If an element or a piece of data is necessary to a critical mission, then it also is critical.

Criticality levels can be assigned to external subjects, data, processes, and devices. Security mechanisms must be put in place to ensure that, commensurate with the criticality level, there is a corresponding confidence that threats against the integrity of the system or its availability cannot succeed (i.e., they are resisted), or if they do succeed, they can be detected; and, if required, lead to a complete and successful recovery of adverse effects. We must protect against deception of mission commanders or users, disruption of mission execution, and usurpation of mission resources. The principal concern in networks is that data arrive accurately, timely, completely and in the same order as transmitted.

Criticality levels are assigned to data, processes or system elements based on a criticality analysis that identifies perils that might befall the mission, taking into account required operational capability and potential threat. Specific operations and operationally critical assets must be identified. Their importance to the mission and their necessity in mission accomplishment are factors. It has been suggested at the New Orleans Workshop [3] that informal models be used to accomplish this, namely mission model, threat model, resources model, and life cycle model.

In a formal integrity model (from Biba [4]), protection level differs from sensitivity as follows: Where the object is data, there is no read access restriction to individuals at lower levels. However, a subject must dominate the object's criticality level in order to originate or modify the data. In the case of processes, the invoking subject must dominate the criticality level of the process. Biba proposed that users and data originating from them carry a set of criticality attributes such that data may be moved only to subjects (humans or processes) bearing an equal or lower level.

APPLICABILITY OF THE ORANGE BOOK

The DoD Trusted Computer System Evaluation Criteria (Orange Book) was developed for use in evaluating trusted commercial components and as guidance for the development and evaluation of trusted computer systems. These criteria are necessary in application to networks and distributed systems, but are not sufficient. Certain terms and concepts (e.g., user and system) must be reinterpreted to adapt to the changing technologies. Other issues include:

- o The interconnected multi-system problem is not adequately addressed in the Orange Book
- o A logical way to deal with the complexity of distributed systems is to divide them into manageable pieces, address security for each of the pieces, and then address the security issues involved in connecting trusted pieces
- o The importance of integrity and denial of service threats to networks
- o By treating sensitivity and criticality as two separate issues we believe that over a range of system implementations, a far more cost effective approach to network security can be achieved.

We see much commonality between the protection criteria for an A division for both criticality and sensitivity, since vulnerabilities and mechanisms have been previously encountered by developers through Orange Book adherence. Many systems have been developed with some criticality protection mechanisms beyond that required by the Orange Book. Off-the-shelf components have not been evaluated against a criticality criteria, but they may still be valuable in a criticality role.

To assist in architectural definitions, we developed a strawman Trusted System Evaluation Criteria (an augmented Orange Book), where the sensitivity level can be specified as before, implying specific mechanisms and levels of assurance (Figure 2, from [5]). Our approach allows independently determining the criticality level required (Figure 3) where the C level concentrates on attack detection, B deals with detection and recovery, and A specifies mechanisms and assurance for resistance, detection, and recovery.

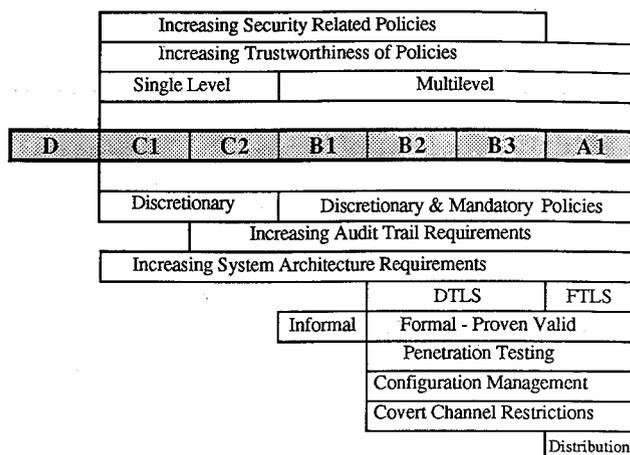


Figure 2.
Trusted System Evaluation Criteria: Sensitivity

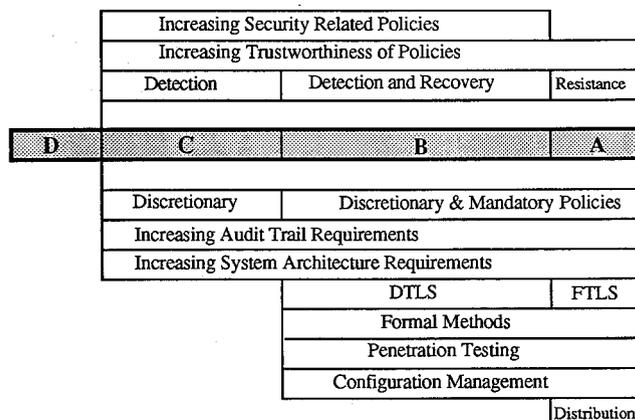


Figure 3.
Trusted System Evaluation Criteria: Criticality

The idea of complete and formal criticality protection is new and untried, at least in the command and control environment. We suspect that due to the opposing nature of the criticality/sensitivity duality, A1/A certifications, for example, will be technologically challenging. By way of an example, in the sensitivity policy we do not want data flowing from a higher level to a lower level. In a criticality policy, flow is allowed, but modification is not.

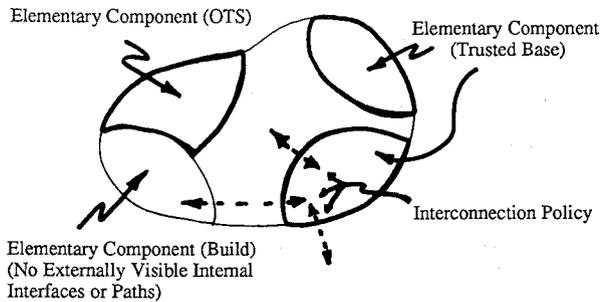
DECOMPOSITION

Evaluation of systems that include networks has three equally important parts: evaluation of externally visible interfaces, evaluation of the internal components, and evaluation of the way in which the components have been interconnected. The Trusted System Base must be well defined and have well specified interfaces. It must include (depend on) the Trusted Base of the components and the Trusted Base previously established for network host systems. Note that this approach has made it unnecessary to precisely define either processor or networks in general.

The Orange Book assumes the system is monolithic, with a single security policy. What is required in a distributed system (the system equivalent to the Trusted Computing

Base) is the assumption that a system is composed of components; in fact there is no part of the system that is not part of a component. The reason for this important deviation from the previous approach is that networked systems may be made up of systems that are either trusted (accredited to handle specific levels of sensitive (or critical) information, where the level of trust is according to some division/class of the trusted system criteria) or untrusted. Different components may have been accredited to deal with different levels and under different trust criteria.

A system is defined by the Designated Approving Authority (DAA) as those physical elements to be accredited as a system (see Figure 4). That decision will be made based on engineering judgement, the scope of authority, and the desire and ability to bring subsystems under a single "umbrella of trust." The system may be geographically isolated or it may be geographically extended. It may be within a physically protected environment or extend through physically unprotected and untrusted areas. The elements may tend to operate autonomously or as a single unit.



Steps:

- Define System Boundaries
- Decompose to Elementary Components
- Determine Interconnection Policy
- Show Components Secure
- Show Aggregate System Secure

Figure 4. System Decomposition

Once the system is defined by the DAA, we can identify what is internal to the system and what is external and interfaces with it. The system security policy must cover all of what is internal, plus the external interfaces. This concept is similar to the Orange Book's use of primary external interface as a human "user." In our Trusted System approach we deal with "external subjects" that may be humans, computers (e.g., hosts), networks, or other systems.

The security policy must identify each of these external subjects. Access control lists may be used to determine what controlled information can be received from them and what controlled information can be sent to them. There must be label consistency or a mapping technique must be defined that ensures proper and complete communication of policy. In some systems it will be necessary to maintain accountability to the user level, even though the user interface is with an external system that interfaces with our system. Sometimes the

policy will require accountability only at the interfacing system level.

For the purpose of illustration, assume that "we" have a system. Sometimes data are passed from one external system through our system to another external systems. Our policy must ensure protection for us and for the external system at the interface. Further, protection may be required by the two external systems to ensure policy compatibility between them, and that is not our responsibility. (An example is when our system is a network that receives encrypted data and delivers encrypted data. If there is a mandatory sensitivity or criticality level separation or a discretionary "need-to-know" or "need-to-modify" exists, appropriate labels and access control lists must be shared between the two external systems communicating data; our network need not necessarily be aware of these requirements or require any action.)

After dealing with external subjects and interfaces we turn our attention inside the system. Our system can probably be decomposed into subsystems and those can in turn be further decomposed. Our goal is to decompose (exactly) to the level at which we have elementary components, where these elementary components in aggregate comprise our entire system. Elementary components may be of three types: 1) Components that are themselves systems and are considered a trusted entity under a single policy (this includes the case where they are untrusted with no policy), 2) components that are off-the-shelf components that have been accredited at some level of trust, and 3) subsystems of the system to be built in which there are no externally visible interfaces or paths and for which a single policy can be determined. Because (at least at the higher divisions) of the required mapping of formal specifications onto an elementary component, it is important that the definition be simple from a security standpoint.

We can now, for security purposes, treat the elementary components as separate systems and, given the security policy of each internally, consider the interconnection policy between these elementary components at their physical interface with each other, at their informational (logical) interface with each other, and with the outside world. We must demonstrate that each of these components is itself a "trusted component" in the sense that its individual security policy is supported, but that it also does not violate system policy. In some cases this demonstration will already have been accomplished through a previous accreditation. For elementary components that must be accredited or reaccredited, this exercise is identical to the process required by the Trusted Computer System (Orange Book) Evaluation. Interfacing components or users are treated as "external subjects."

Now we look at security policy from the system level. At this level it must be assured that all component policies are supported throughout the system (including data that eventually are passed between

components that do not interface directly). Further, there may be policy dictated at the system level that is over-and-above the policy that exists at the individual component level and it must be ensured that policy is supported. Finally, there exists a policy at the system level as to its interface with the outside world, and it must be ensured that this system level policy is supported by the components that interface with the outside world (e.g., external subjects).

When a component is upgraded, decomposition can be used to reduce network reaccreditation costs. Decomposition can also be used when subsequent expansion (components), concatenation (two or more systems plus a gateway), and extension (addition of protocol layers beyond those presently implemented) of the system occur.

SENSITIVITY AND CRITICALITY THREAT

An excellent discussion of sensitivity threat and mechanisms can be found in Voydock and Kent [6]. In studying sensitivity and criticality threats, it is discovered that they differ significantly (see Figures 5, 6, and 7). Treating sensitivity and criticality separately may have an economic advantage in that when data are stored and being communicated, their sensitivity can be protected with encryption. Once that is accomplished, we can address criticality.

PROTECTION MECHANISMS

The mechanisms employed against sensitivity and criticality attacks depend to a great degree on the protection environment afforded by physical protection, and the clearance and access controls in place (Figure 8). If encoding is used as a mechanism, the distance (link or node) must be determined, or that part of the total system over which it is employed. Finally, although resistance is the primary choice for sensitivity protection, detection, as well as recovery must also be considered for criticality or for the protection of the integrity of sensitivity mechanisms.

Sensitivity mechanisms are generally known so are not itemized here. For this paper, it is important to identify mechanisms employed against the criticality threat. These are presented in Figure 9, itemized according to whether detection, recovery, or resistance is the objective.

SECURITY CONSIDERATIONS

Traditional security factors considered in monolithic computer systems also apply when developing a distributed security architecture. In addition, researchers and practitioners in network security have identified several factors that must be addressed if we are to achieve an acceptable solution to the network security problem. This section reviews those factors.

The N-Squared Problem

The N-squared problem refers to the

Release of Message Contents

- Leaking
- Inference
- Browsing
- Crosstalk

Protection Mechanism Integrity

- Modify Data
- Overwriting

Spurious Association Initiation

- Masquerading

Figure 5. Sensitivity Attacks

Traffic Analysis

- Message Lengths
- Transmission Frequencies
- Source and Destination

Message Stream Modification

- Authenticity (Misrouting, Inserting, Replaying)
- Integrity (Modifying Data, Overwriting)
- Ordering (Deleting, Duplicating, Altering order of data or block)

Denial of Message Service

- Discarding Messages
- Delaying Messages
- External Physical Attack

Spurious Association

- Masquerading

Figure 6. Criticality Attacks

Sensitivity:

- Continuous window
- Information stolen for enemy exploitation
- Detection difficult, no recovery once gone
- Sophisticated agent data processing/communications attack

Criticality:

- Malicious (up to creating an incident) continuous window
- Nullifying crisis response has a smaller window during crisis
- Sophisticated agent for integrity attack
- Nonsophisticated agent for denial of service threat
- Chance to recover exists, based on detection mechanisms and availability/survivability design

Figure 7. Attack Characteristics

Physical Security Affects Choices

Mechanism Application Point

- Link Level
- End-to-End
- Association Level

(Mechanism Approaches)

Resistance

- Trusted Functionality

Detection

- Automated Auditing
- Local and/or Remote Reporting

Recovery

- Unilateral Disconnection
- Repeat Action
- Alternate Action

Detection Mechanisms

- Modification Detection Codes (Message Integrity Codes, Message Authentication Codes)
- Critical Mission Model
- Utilization Statistics Model
- System Failure Model
- Security Feature Denial of Service

Recovery Mechanisms

- Isolation
- Repeating Process
- Changing Process
- Survivability (Adaption, Self Healing, Backup and Fault Tolerant)

Resistance Mechanisms

- Reference Monitor
- Traffic Analysis Prevention (Spurious Traffic, Message and Traffic Padding)
- Survivability Strategies

Figure 8. Mechanisms in General

Figure 9. Criticality Mechanisms

complexity that must be considered in access control. In the extreme, each individual user must know the identity and access characteristics of each of the other users (including himself), hence the name.

We have historically performed document access control based on a so called hierarchical N-squared system. At an office level each person must know the authorization of each person in that office because they are in close proximity. Documents are passed between offices through a local security officer. At higher organizational levels, documents are again passed between organizational security offices. At an agency, corporate, or service level, both documents and security clearances/authorizations are passed. At a national level, clearing agencies exchange information. That is, an N-squared problem is addressed at each level of the hierarchy, but only with the elements at that level.

Bridges and gateways can link networks. Networks or linked internets can connect individuals, organizations, or agencies. As a starting point, information can be controlled in the historical way. However, the power of data processing allows the number of hierarchical levels to be reduced and perhaps eliminated completely, thereby dealing with the N-squared problem.

The Cascading Problem

The networking of systems introduces the cascading problem (see [7]), which is the increase in exposure (the range between the highest classification of data and the lowest user clearance) in interconnected systems. For example, if a TOP SECRET/SECRET system passes only SECRET data to a SECRET/CONFIDENTIAL system, the TOP SECRET data has now been exposed or contaminated at the CONFIDENTIAL level instead of just the SECRET level. As more and more interconnections are made, in general, the highest level will be exposed at the lowest level. Therefore, either all high level data must be protected at a high (A1 or B3) protection level or more secure (but less flexible) modes of operation (e.g., dedicated or system high) must be used.

The Security Policy Problem

For each of the entities at each of the sensitivity and criticality levels in the hierarchy, different policies might exist (primarily because of different threat, mechanisms, and objectives). For a trusted base (or the untrusted protection equivalent), policy is a statement (mathematical or formally written) of security motivated constraints (such as discretionary and mandatory access controls). These are the constraints to be placed on the modification and/or dissemination of data (including control data); the initiation, control, or termination of processes; and/or the assignment or use of system resources. Policy mapping (see [8] for examples of policy mapping) is the establishment of a common interconnection policy between two communicating entities, each with inherently different policies. It identifies legal communications, communications constraints,

required labels, required transformation of labels from the form used by the sending system to that used by the receiving system, and an agreement as to mechanisms to be used and their placement in the the communicating systems.

Security Models

As discussed by Crosland and Schnackenberg [9], the distribution of security functions and features across a network complicates the system design and formal specification. In centralized systems, TCB requests are mediated by a single component, and thus can reasonably be represented by a single state transition. However, for a network the trusted base is distributed and disjoint, so that actions at one trusted base interface affect remote trusted base state and remote trusted base interfaces. For example, when a host or terminal user requests a connection for a session, the local trusted base software coordinates with the remote trusted base supporting the destination device, and possibly with network management to determine if the session is authorized. The states of two or three trusted bases are changed as a result of a new session being created and the session creation event is visible at the external interfaces of two trusted bases. Thus, a single TCB request can cause the distributed Trusted System Base (TSB) to undergo multiple state transitions. There are two approaches that can be used: a) ignore the concurrency and distribution of functions, and treat state transitions as if they all occurred atomically or b) describe the interaction between the remote TCBs.

We have taken the latter approach with a hierarchical modeling methodology. First, model each elementary component using the techniques developed for centralized systems. Then model a system composed of components dealing with only the subjects and objects visible in the external communications. Reducing the complexity allows modeling state transitions. When the system itself becomes an elementary component, this process is repeated. Mechanisms similar to deadlock avoidance in "association" mechanisms assure the absence of mutually conflicting security state transitions.

Covert Channels

The covert channel analysis problem is also discussed in Crosland and Schnackenberg [9]. In a stand-alone system the covert channels tend to be between processes under the control of an operating system. In a network, however, there may be few interprocess covert channels. This is due to the limited resources available to processes that reside within the network servers. The major covert channels are between processes that reside in attached hosts and workstations, and signal each other using network resources. Although the network can detect possible usage of this covert channel, the network is not able to reasonably eliminate it. The host along with the host front end has the responsibility to restrict access to (or close) the covert channel.

Protocol Issues

The International Standards Organization Open System Interconnection (ISO/OSI) seven layer protocol reference model [10] has gained wide acceptance, unfortunately however, not before the Government had already drawn up some very firm procedures for handling data in communications and networks. The DoD has made a commitment to move in the direction of the seven layered approach in its future planning and development, while at the same time ISO has begun to deal with some of the sticky problems that are typical to the DoD applications (e.g., security).

Figure 10 illustrates the functions present at various layers and how intercommunication of these functions actually takes place at the next lower layer. The higher level protocols are present at the communicating nodes where the applications reside. The communicating nodes and devices within the network itself communicate to one another through the first three layers.

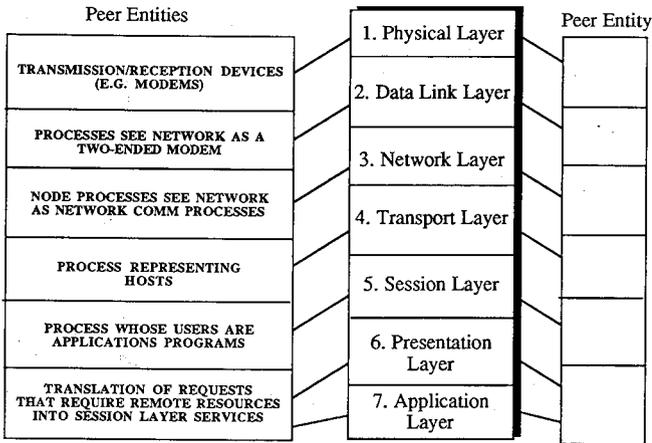


Figure 10. The ISO/OSI Protocol Reference Model

Figure 11 illustrates the potential security implementations as proposed by the ISO Draft Security Model [11]. End-to-end encryption can be accomplished in the network layer (3), the transport layer (4), or the presentation layer (6). If there is a choice, the higher the layer, the greater the protection. In the protocol traffic analysis problem, if no mechanisms were employed, it would be desirable to do the end-to-end encryption at the network layer (3) first, the transport layer (4) second, and finally the presentation layer (6). (Note that the session layer (5) as defined by the ISO Reference model will not support encryption.)

Network service requests might very well be covert channels and therefore one would want to minimize network services by interconnecting trusted bases where only routing was required or enforce a limited bandwidth in the use of those services. The Draft OSI Reference Model on Security proposes the availability of many network services by which a user can employ security or ignore it. This is contrary to the DoD idea of having continuous security protection mechanisms in force that have minimum

interference with mission operations.

Figure 12 depicts an approximate relationship between the ISO model and the commonly accepted protocols inherent to DoD communications. The lack of strict compatibility of layers at level 3 and above is illustrated here, but in fact varies, not only in people's minds, but from application to application. Summarizing the primary differences, DoD has historically divided the network layer into sublayers and in addition, there have not been well defined layers above the host-to-host interface.

Mechanism	Layer						
	1	2	3	4	5	6/7	
Confidentiality	x	x	x	x	*		
Access Control			x	x	*		
Peer Entity Authentication			x	x	*		
Origin Authentication			x	x	*		
Nonrepudiation (Origin/Delivery)					*		
Criticality				x	x	*	
Traffic Flow Security	*	*				*	

(x and * = ISO possible implementations,
* = Ideal from our perspective)

Figure 11.

Security Implementation by Protocol Layer

ISO Model	Corr DoD Function	DoD Protocols	ISO Equivalent
7. APPLICATION	Process/Applications	FTP, SMTP TELNET (Native Mode)	FTAM, X.400 VTP Terminal
6. PRESENTATION			
5. SESSION	Host/Host	TCP, TACACS, UDP	TP
4. TRANSPORT	Internet	IP, ICMP, HMP	ISO - IT
3. NETWORK	Network	X.25 Long Haul Arpanet IEEE 802	
2. DATA LINK	Data Link Control	ADCCP HDLC, X.25, BBN1822	
1. PHYSICAL	Physical	RS232C, MILSTD 188C MILSTD 188-114 RS422A, 423A, 449	

Figure 12. DoD Protocols

The specific standards written for military use are addressed in the third column of Figure 12. This is a mixture of civil standards and military specifications. The military is migrating to the civilian X.25 and IEEE 802 standards, while at the same time commercial versions of TCP and IP exist in the marketplace. ISO equivalent standards (illustrated by the far right column) are striving to encompass the features and characteristics of the equivalent military protocols while maintaining a strict adherence to their model. The DoD has said that if the ISO efforts are successful, DoD will eventually adopt the ISO models.

ARCHITECTURAL APPROACH

Based on the sensitivity and criticality requirements in mission-critical networks,

and considering the above discussions of threats, mechanisms and protocols we have developed a functional description for secure networks. The functions are highlighted in this section.

The protocol layer choices we have committed to in our architecture are presented in Figure 13.

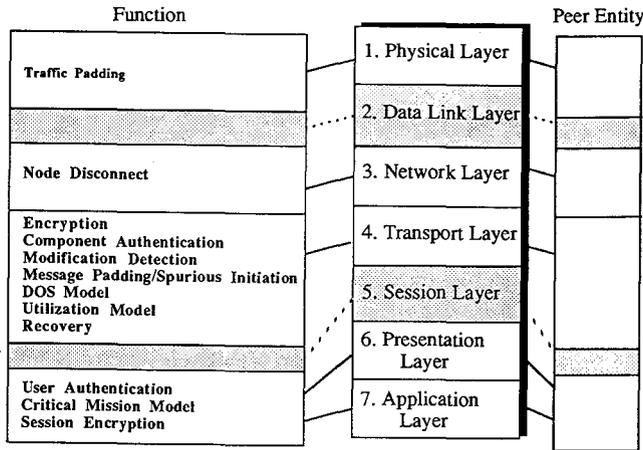


Figure 13. Protocol Layer Choices in this Architecture

Encoding Replaces Physical Protection

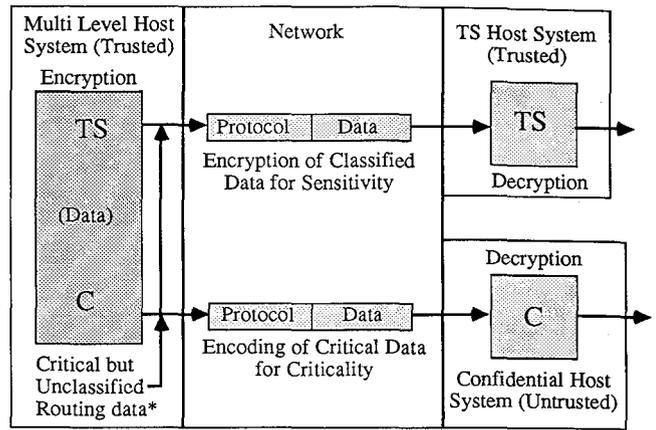
Cryptography has long been a major COMSEC protection mechanism where it is known that physical protection cannot be provided. Today's VLSI circuitry can provide encryption protection that is potentially transparent to the transmission and storage processes, and does not impact performance.

Further, application of encoding encompasses much more than simple encryption. With cryptographic checksums (seals) and other mechanisms, modification and replay can be detected. With public key approaches, senders and receivers can be authenticated, and even the precise source of a message can be guaranteed days or years later.

Even within physically protected areas, a strong mechanism against internal attack is use of encoding for both sensitivity and criticality protection. Crypto checksums for criticality can be used for all data at all times. Encryption of sensitive data can be employed at all times except during computation on the data or its human input/output.

Such emphasis focuses the burden on covert channel elimination and/or protection and detection mechanisms, since such an encoding approach for sensitivity requires that certain levels of protocol remain in the clear.

Figure 14 shows our architectural approach to networks, using these concepts. To solve the sensitivity problem, mission data are encrypted. To solve the criticality problem, all data are encoded using cryptographic checksum and authentication. The covert channel problem of data leakage through header information is addressed in the host systems rather than in networks.



* Covert channel problem must be addressed in host systems

Figure 14. System Security Strategy

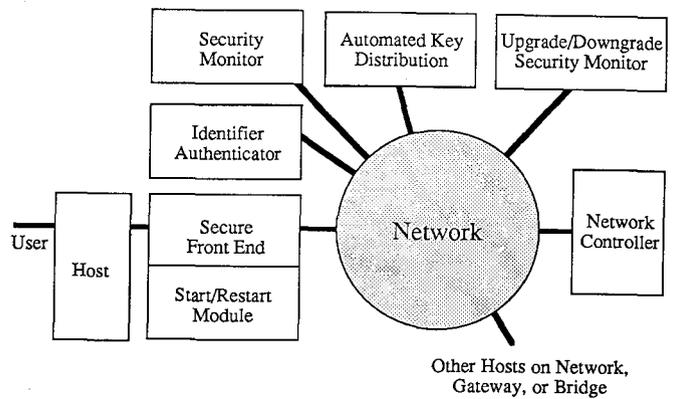


Figure 15. Distributed Security

Distributed Security Mechanisms

Figure 15 illustrates an extension to an important concept developed for the Blacker program in which elements of the security systems are themselves nodes on the network. In the Blacker approach [12] there is a front-end node for each of the system hosts and internetwork gateways, and in addition there are nodes for a security monitor position, a centralized key distribution function, and a central identification/authentication database.

We have added additional functions including an upgrade/downgrade position to deal with high risk communications and to act as a resource for use between nodes of differing security policies. Also, a network control function establishes secure communication through gateways and across network bridges. Although shown in the figure with the secure front-end, all elements depend on a hand-held start/restart module when first coming up on the network or when being removed, so security is not violated. This was another important concept implemented by the Blacker program.

Distributing these capabilities allows expansion of the network at minimal security cost and impact. Backup security functions are facilitated, since each capability can exist redundantly on the network. It also allows adaptation to the load, for example, where several upgrade/downgrade monitor

positions can exist to keep up with the traffic in a high risk environment.

Multi-Risk Internet Communication

In our proposed architecture we needed to reduce the risk of interdomain communications where high exposure and/or high-risk connectivity potentially exists. A concept was proposed in which three modes of connectivity are established and supported by the access control function (see Figure 16):

- o A direct trusted exchange to low risk domains controlled only by the access control rules
- o An exchange where extra-domain authentication must be performed manually by the security monitor prior to allowing an association in medium risk connections
- o A monitored and verified exchange by a manual (human) guard in an upgrade/downgrade monitor position for high risk connections

- Assign Domain Vulnerability
- Adaptive Mechanism Based on Trust
- Depends on One-Way vs Two-Way Communications

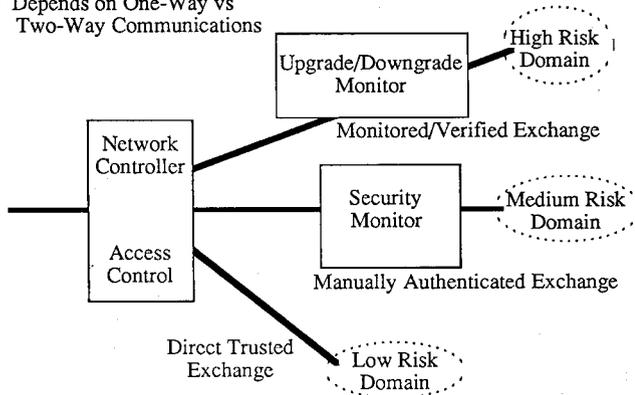


Figure 16. Multi-Risk Internet Communication

Association Level Services

In Blacker, once data are delivered to the host, protection ceases to exist unless provided by the host. We have proposed an approach in our architecture that has, at a minimum, the Blacker level of protection, and for sensitivity and/or criticality, protection all the way to the device interfacing with the user. This association level protection (Figure 17) provides key distribution for sensitivity encryption, criticality encoding, identification, and authentication right up to the microprocessor that interfaces with the user, assuming the appropriate enciphering hardware is present. A communication, though initiated through a host, can be protected from that host and its other users. Associated chips and/or algorithms must be contained in the microprocessor. All security services (security monitor, identification authentication, key distribution, etc.) within the network become part of this association level protection.

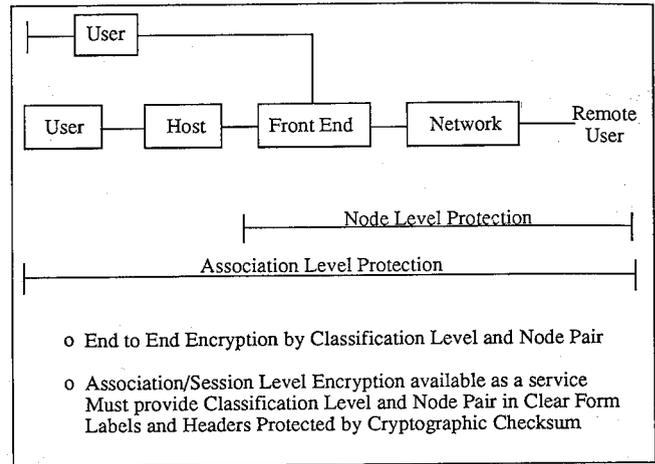


Figure 17. Association Level Services

SUMMARY

The proposed approach to network security outlined in this paper separates the sensitivity requirement (protection of classified information) from the criticality requirement (integrity of operations and protection against denial of services). This decision has resulted in the ability to use encryption and covert channel protection mechanisms to solve the sensitivity problem in host communications and data storage problems, leaving only the criticality problem to be addressed. However, the criticality problem can generally be solved in networks with detection and recovery approaches (existing primarily in the host protected domain) which are far less costly than resistive (formal model) mechanisms.

For interconnected hosts/networks, we have found that differences in security policy and different levels of risk may be confronted head-on by means of decomposition. Increased exposure must be considered in assessing and determining required protection levels. Interface policy must be established and supported both from a mandatory and a discretionary perspective. The reference monitor concept must be used to control access at the network, component, and individual user levels.

We have proposed architectural concepts for computer networks that emphasize standardization, shared functions, and operation with planned networks. Our solution uses end-to-end protection for criticality and sensitivity with association level protection as an added service. The proposed functions to be performed (a superset of Blacker functionality) include security monitor, identification authentication, key distribution, network control, upgrade/downgrade, and start/restart.

ACKNOWLEDGEMENTS

This effort began with a complete survey of the literature on network security, especially the results of the New Orleans Conference [3]. Most of the concepts presented here are a result of choosing a compatible set of the the ideas resulting from that conference, in combination with

much of the work that was done associated with Blacker as well as the model of Biba. In addition to papers referenced in the text, we have borrowed ideas from C. Meyer and S. Matyas [14], G. Popek and C. Kline [15], M. Schaefer and D. Bell [16], D. Denning [17 & 18], and S. Walker [19]. We wish to thank D. Branstad for review and assistance.

REFERENCES

1. DoD 5200.28-STD, "Trusted Computer System Evaluation Criteria," December, 1985
2. AFR 205-16, Automatic Data Processing (ADP) Security Policy, Procedures and Responsibilities, Department of the Air Force, August 1, 1984
3. Brand, S.L., ed., "Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security," New Orleans, LA, March 19-22, 1985
4. Biba, K.J., "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA, April 1977
5. Kaiser, W.G., "The Making of a B2 System," Proceedings, 1986 AFCEA Symposium on Physical/Electronic Security, pp. 21-1
6. Voydock, V.L., and S.T. Kent, "Security Mechanisms in High-Level Network Protocols," ACM Computing Surveys, Vol. 15, No.2, June 1983, pp. 135-171.
7. Millen, J.K., "A Network Security Perspective," Proceedings, 9th National Computer Security Conference, NBS/NCSC, 15 September, 1986, pp. 7-16
8. La Padula, L., "Some Thoughts on Network Security: A Working Paper," Proc. DoD Computer Security Center Invitational Workshop on Network Security, DoD Computer Security Evaluation Center, Ft. Meade, MD, March 19-21, 1985, pp. 2-49, 2-60
9. Crosland, M. and D. Schnackenberg, "Application of Formal Techniques to a Multilevel Secure Local Area Network," Proceedings, Symposium on Physical/Electronic Security, August 1986, Philadelphia Chapter AFCEA, pp. 2-1 thru 2-5
10. International Standards Organization, Draft International Standard 7948: Data Processing - Open System Interconnection - Basic Reference Model, 1983
11. N1925, International Standards Organization, Working Draft Addendum to ISO 7948 to Cover Security Architecture, ISO/TC 97/SC Copenhagen, June 19-28, 1984
12. USG Memo, "Computer Security Certification Plan for BLACKER Phase 1," Appendix D to Purchase Description C5-001, Blacker COMSEC Development, Computer Security Certification Requirements, October 18, 1984
13. "Department of Defense Trusted Network Evaluation Criteria, DRAFT," July 29, 1985
14. Meyer, C.H., and S.M. Matyas, Cryptography: A New Dimension in Computer Data Security, John Wiley and Sons, New York, 1982
15. Popek, G.J., and C.S. Kline, "Encryption Protocols, Public Key Algorithms, and Digital Signatures in Computer Networks," in Foundations of Secure Communication, ed. by R. A. DeMillo, et al, Orlando, FL: Academic Press, 1978, pp. 133-154.
16. Schaefer, M., D.E. Bell, "Network Security Assurance," Proceedings of the 8th National Computer Security Conference, Gaithersburg, MD., September 30, 1985, pp. 64-69
17. Denning, D.E., "A Position Statement on Network Security," Proc. DoD Computer Security Center Invitational Workshop on Network Security, DoD Computer Security Evaluation Center, Ft. Meade, MD, March 19-21, 1985, p4-47,4-56
18. Denning, D.E., Cryptography and Data System Security, Addison-Wesley, 1982
19. Walker, S.T., "Network Security Overview," Proceedings 1985 Symposium on Security and Privacy, IEEE Computer Society, April 1985, pp. 62-66
20. CTA, "Draft AFSPACECOM Trusted System Evaluation Criteria," NCCS-86-03-383, March 1987

A SECURITY MODEL AND POLICY FOR A MLS LAN

Peter Loscocco

Office of Research and Development
National Computer Security Center

I. BACKGROUND

The multilevel secure local area network (MLS LAN) to which this security policy and model apply is a broadband cable bus LAN that uses Transmission Control Protocol/Internet Protocol (TCP/IP) and Carrier Sense Multiple Access (CSMA/CD). The LAN is capable of having hosts that range from single-level, untrusted machines to MLS systems with classified and compartmented data. Every host on the LAN will be connected to the bus via a Bus Interface Unit (BIU).

The LAN is still only in the design stage. As the design changes, the security policy or model may also require changes. Both were written with this in mind and should be flexible enough for most situations. As it stands now, the model does not totally describe some aspects of communications on the LAN. These shortcomings have been noted and will be corrected in future versions.

II. SECURITY POLICY

The MLS LAN will implement a security policy based on the Department of Defense (DoD) Security Policy [1]. That policy states that a person, or machine, may not be granted access to classified data unless that person, or machine, has the proper security clearance and has a need to know that data. There are also provisions for special handling restrictions (or caveats) to be added to data, these restrictions must be obeyed whenever the data is accessed.

In the context of the MLS LAN, this policy pertains to the BIU sending and receiving a packet. All data on the LAN is transmitted in the form of a packet. A packet contains the data to be communicated as well as that data needed to deliver it. This includes everything from addresses and other header information to the security label. There are five basic rules the LAN must enforce to assure the DoD policy is followed.

- Rule 1: Packets on the LAN must be properly labeled to reflect their security level.
- Rule 2: A BIU may not transmit a packet unless it is authorized to do so.
- Rule 3: A BIU may not deliver a packet to a host unless it is authorized to do so.
- Rule 4: Packets delivered to a host must not have been altered.
- Rule 5: All security-related events must be logged to provide an audit trail of any security violations that might occur.

Rule 1 states that all packets must have a security label while they are within the security perimeter of the LAN. Within this perimeter are the BIU's, the cable bus itself, and a special host called the access controller (AC) whose function will be explained below (see Figure 1). The security label must correctly reflect the packet's classification and any compartments or handling restrictions that might apply. Clearly, many of the BIU and AC functions must contain a high level of trust.

It is the job of the BIU to enforce Rule 1. The BIU must always know the certified level of trust of the hosts to which it is attached. It must also know the security level

of its current connections. When the BIU receives a packet from a host, it will first check the level of trust of the host. If the host's level of trust checks, the BIU will use the security label provided by the host. If not, the BIU will assign a label that reflects the level of the current connection.

Rules 2 and 3 together state that all communication with the LAN will be in accordance with the DoD security policy. Rule 2 prevents a BIU from transmitting data from a host whose security level is either too high or too low. It also assures that a packet from a host only gets sent to hosts who are cleared and have a need to receive it. Rule 3 prevents a BIU from delivering packets to an attached host who has neither the required clearance or need to know. Furthermore, this rule allows hosts to have a minimum security level placed on them for incoming packets and guarantees that packets are only delivered to the hosts to whom they are sent.

A BIU can only enforce Rules 2 and 3 if it is able to make decisions on whether or not to send a packet to, or receive a packet from, another BIU based on the address of that packet, its security level, and the clearance and need to know of each of the BIU's. The security label of a packet identifies its security level and must correctly reflect the packet's classification and any applicable compartments or handling restrictions that might apply. A BIU's clearance and need to know are determined from access control tables.

The access control tables contain the mandatory and discretionary access control (MAC/DAC) [2] information for each BIU and BIU pair. They reside on the AC and are set up and maintained by the Network Security Officer (NSO), the only user permitted to actually sign onto the AC. The NSO is responsible for ensuring that each host's entries in the tables properly reflect the security levels, compartments, and handling restrictions of data that reside on that host. He is also responsible for ensuring that the tables properly reflect which hosts can communicate at what levels to provide which services.

To start communicating, one host (H1) would send a request addressed to another host (H2) specifying the security level and type of connection wanted. H1's BIU will recognize this as a connection request and reroute it to the AC. Based on the access control tables, the AC will determine whether the connection should be approved. H1

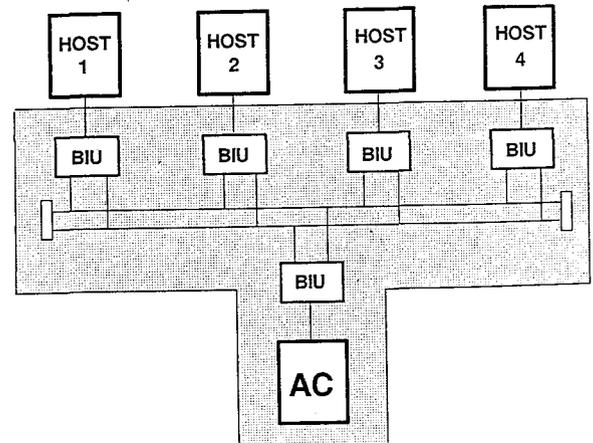


FIGURE 1: SECURITY PERIMETER

is notified if the connection is not in accordance with the MAC/DAC policy, and H2 is not contacted. If the connection is in accordance with the policy, however, the AC sends the request to H2 for approval or disapproval. H2 then sends either a connection acceptance or rejection addressed to H1. However, H2's BIU reroutes this back to the AC. If the connection is to be opened, the AC logs the opening in a table, notifies H1, and instructs the two involved BIU's to set their current connection status to reflect the proper hosts and levels.

The security of the connection now rests with the BIU. The AC is not contacted again until the connection is closed or a security-relevant event occurs. A packet reaching a BIU, either from one of the hosts or the LAN, is accepted or rejected according to the levels of the connection as set by the AC. In this way, the LAN guarantees that only authorized packets enter and leave the security perimeter.

Rules 2 and 3 cannot be properly enforced without Rule 4, and both depend on communications with the AC. It is imperative that these packets not be tampered with because unauthorized connections could otherwise occur. Fortunately, Rule 4 can be enforced using the proper authentication and encryption techniques.

Rule 5, strictly speaking, will not increase the security of the LAN. Rather, it is included to increase the confidence that the LAN is secure and the probability that a security breach will be detected and the responsible party(ies) identified.

The auditing capabilities of the LAN will be in the BIU and the AC. The BIU will report to the AC, and the information will be stored there for later review by the NSO.

III. SECURITY MODEL

This model is a mathematical description of the secure operation of the MLS LAN. A model of the LAN must include three separate things: a BIU, the AC, and the communications between a collection of BIU's and the AC. The operation of the LAN is said to be secure if the five rules given above are being enforced at all times.

The model is in two parts. The first part introduces some concepts and functions needed to mathematically restate the rules given above and ultimately does so. Some of the concepts were borrowed from the model specified by the Worldwide Military Command and Control System (WWMCCS) in "The Formal Model for Secure Data Distribution in the WWMCCS Information System (WIS)."[3] These concepts have been modified to reflect the actual differences between the operations of a system-high network, such as WIS, and a truly multilevel network with hosts of varying security levels. The second part of the model describes the BIU's and the AC with a system of intercommunicating state machines.

A. Mathematical Restatement of Security Rules

Before the rules can be stated mathematically, some definitions need to be introduced. The security labels of Rule 1 have to be formally defined. Four functions are required to describe Rules 2 and 3. These functions are: send-packet, receive-packet, connect-open, and connect-close. To be complete, one must postulate the existence of two more functions, unaltered and audited, that guarantee the enforcement of Rules 4 and 5, respectively.

Assume there is a set, P, of packets which contains all the potential packets on the LAN. Rule 1 dictates that a classification level must be assigned to each p, an element of P, to identify its security level. This label can be described as a 3-tuple as follows:

$$\text{Level} = (S, C, H)$$

where

S = sensitivity level,
C = compartmented information set, and
H = handling restriction set.

It is the sensitivity level, S, which indicates the data's classification. The range of possible values for S come from a set, ES. There exists a ranking function, R, which places a definite ordering on the elements of ES. The possible sensitivities, as ordered from lowest to highest by R, are: Unclassified (U), Encrypted For Transmission Only (EFTO), Restricted (R), Confidential (C), Secret (S), Top Secret (TS), and Program and Control (PROG).

It is the compartment set, C, that contains the need-to-know access control information. All possible elements of C are drawn from a set, EC. Unlike ES, this set is not hierarchical. Each element, Ci, represents a compartment into which a given data unit can be placed. A null C represents data which is not compartmented.

The handling restrictions set, H, also draws its elements from a nonhierarchical set, EH. As its name implies, this set contains a set of restrictions which must be adhered to when handling a given data unit. As with C, a null H represents no handling restrictions.

All possible data security levels come from what is called the Classification Set Space, denoted C-Space. C-Space is derived from the three sets: ES, EC, and EH. It is defined as the Cartesian product:

$$\text{C-Space} = \text{ES} \times \text{P}(\text{EC}) \times \text{P}(\text{EH})$$

where P() represents the power set or set of all possible subsets of the respective sets.

A partial ordering of C-Space can be achieved by introducing the concept of security dominance. Given any two security labels, Lx and Ly, such that

$$Lx = (Sx, Cx, Hx) \text{ and } Ly = (Sy, Cy, Hy),$$

Lx is said to be dominated by Ly if and only if

$$\begin{aligned} R(Sx) &\text{ is less than or equal to } R(Sy), \\ Cx &\text{ is a subset of } Cy, \text{ and} \\ Hx &\text{ is a subset of } Hy. \end{aligned}$$

Let there be a function, dominate(Lx, Ly) where Lx and Ly are elements of C-space, which returns true if and only if Lx dominates Ly.

Let there be two functions, label(p) and s-label(p,l) where p is an element of P and l is an element of C-space, that read a label from, or set the label of, a packet.

Several concepts and functions need to be introduced before send-packet, receive-packet, connect-open, and connect-close can be defined.

There exists a set, B, defined as:

$$B = \{ b \mid b \text{ is a BIU on the LAN} \}.$$

B is necessarily nonempty. It must at least contain an element, B-AC, which represents the AC's BIU.

Let there be a function, id(), that returns a b, an element of B, which is the BIU that executed the function. This function allows a BIU to determine its own identity.

Two functions exist, mode and s-mode, which are defined as follows:

mode(b) - returns current operating mode of b, an element of B
0 if packet labels from the attached host can be trusted
1 if packet labels from the attached host cannot be trusted

s-mode(b, m) - sets current BIU operating mode
 b = BIU to be set - an element of B
 m = 0 or 1
 0 if labels from host are to be trusted
 1 if labels from host are not to be trusted

A BIU's operating mode must be either zero or one. Authorization for s-mode to be executed may only come from the AC.

For each b, an element of B, there is an access control set (ACS). ACS's reside on the AC and are uniquely identifiable by b. The ACS contains all of the MAC information that the AC will need to determine if b's participation in a given connection will violate the MAC policy. Mathematically, an ACS is a subset of ACS-space defined as the Cartesian product:

$$\text{ACS-space} = (T \times \text{AI} \times \text{AI})$$

where T is the set of all connection types and AI is the set of all access intervals. T and AI are defined as follows:

$T = \{ t \mid t \text{ is a connection type} \}$ and
 $\text{AI} = \{ (a1, a2) \mid \{a1, a2\} \text{ is a subset of } c\text{-space and } \text{dominate}(a2, a1) \}$.

At present there are only four elements in T. They are remote access, R; file transfer, F; mail, M; and control, C. Type C is reserved for communication with the AC. As the need arises, more elements may be added without effecting the model.

The components of each access interval are the minimum and maximum security levels that a packet belonging to a particular connection may be and still pass through the BIU. The second and third components of each element of the ACS represent the two directions, going out of and coming into the host, that packets may flow through a BIU. Each element of an ACS represents a different range of security levels at which a given host may participate in a connection of a given type. In practice, the minimum level of all incoming packets will usually be (U, {}, {}).

Two functions exist, min() and max(), which take an access interval as an argument and return its respective minimum and maximum security levels. They are defined as follows:

if (a1, a2) is an element of AI,
 then $\text{min}((a1, a2)) = a1$ and $\text{max}((a1, a2)) = a2$.

For each b, an element of B, there is also a discretionary access set (DAS). DAS's also reside on the AC and are uniquely identifiable by b. The DAS contains all of the DAC information that the AC will need to determine if b's participation in a given connection will violate the DAC policy. Mathematically, a DAS is a subset of DAS-space defined as the Cartesian product

$$\text{DAS-space} = (B \times T),$$

where B and T are as above. If a BIU, b1, has a DAS that contains an element (b2, t), discretionary access of type, t, to BIU, b2, could be granted to b1.

With the ACS's and DAS's, the AC has all of the necessary access control information to ensure that the security policy is not violated. The NSO must take great care in specifying the ACS's and DAS's to insure that the MAC/DAC policies are properly enforced.

Two functions are defined to describe the access checking done by the AC for one BIU. These functions, mandatory-access and discretionary-access, are as follows:

mandatory-access(b1, type, aio, aii) and
 discretionary-access(b1, b2, type),

where

b1 = the BIU for which the checking is being done - an element of B,
 type = the type of connection in question - an element of T,
 aio = the outbound access interval of the connection in question - an element of AI, and
 aii = the inbound access interval of the connection in question - an element of AI.

Mandatory-access(b, type, aio, aii) returns true if and only if there exists an element of b's ACS,

$a = (t, (\text{min}1, \text{max}1), (\text{min}2, \text{max}2)),$

such that

t = type,
 dominate(min(aio), min1),
 dominate(max1, max(aio)),
 dominate(min(aii), min2),
 dominate(max2, max(aii)).

Discretionary-access(b1, b2, type) returns true if and only if there exists an element in b1's DAS,

$a = (b, t),$

such that

b = b2 and
 t = type.

Using mandatory-access and discretionary-access, it is possible to more completely describe what is meant by an authorized connection. A connection of a given type may be authorized between two BIU's at given access intervals if the mandatory and discretionary access checks succeed for each host. A new function, open-ok, returns a true value when it is possible to authorize a connection. It is defined as follows:

open-ok(b1, b2, t, ail, ai2),

where

{b1, b2} is a subset of B,
 t is an element of T, and
 {ail, ai2} is a subset of AI,

returns true if and only if

mandatory-access(b1, t, ail, ai2),
 discretionary-access(b1, b2, t),
 mandatory-access(b2, t, ai2, ail), and
 discretionary-access(b2, b1, t).

It is important to note that a return value of true here does not mean that a connection has been established between b1 and b2 but only that such a connection would not violate the MAC/DAC policy.

A packet may exist on the LAN only if it was transmitted through an authorized connection. In managing all of its host's connections, a BIU assigns a currently unassigned connection number to each connection it establishes for its host. It is important to note that the two BIU's involved in a connection may refer to that connection with a different connection number. These connection numbers are elements of a set, Connections, denoted by CN. Each of a BIU's connections may be uniquely identified by an ordered pair, (b, cn), where b is an element of B and cn is an element of CN.

Let there be a function new-cn(b), where b is an element of B, that assigns an unused connection number to the BIU, b. This function is used in the opening of connections.

There is certain information kept at every BIU for each possible connection. This includes the connection type,

the other BIU involved, and the access intervals. The following functions exist to retrieve this information:

ct(b, cn) - returns the current connection type
t : an element of T if the connection exists
NULL : if there is no connection

cb(b, cn) - returns the current BIU connected to
b1 : an element of B if the connection exists
NULL : if there is no connection

ccn(c, cn) - returns the connection number used by the other BIU
cn1 : an element of CN if the connection exists
NULL : if there is no connection

caio(b, cn) - returns current outbound access interval
ai : an element of AI if the connection exists
NULL : if there is no connection

caii(b, cn) - returns current inbound access interval
ai : an element of AI if the connection exists
NULL : if there is no connection

where

b = the BIU in question - an element of B and
cn = the connection number in question - an element of CN.

Five functions exist to set this connection status information in a BIU. Each of these functions has three arguments: the BIU, the connection number, and the information to be set. They are executed exclusively at the request of the AC and return true if and only if the information is properly stored. The five functions are defined as follows:

s-ct(b, cn, t) - sets the current connection type
b = BIU to be set - an element of B
cn = connection number on the BIU to be set - an element of CN
t = new connection type - an element of T or NULL

s-cb(b1, cn, b2) - sets the current BIU connected to
b = BIU to be set - an element of B
cn = connection number on the BIU to be set - an element of CN
b2 = other BIU - an element of B or NULL

s-ccn(b, cn1, cn2) - sets the connection number of the other BIU
b = BIU to be set - an element of B
cn1 = connection number on the BIU to be set - an element of CN
cn2 = connection number on the other BIU - an element of CN or NULL

s-caio(b, cn, ai) - sets the current outbound access interval
b = BIU to be set - an element of B
cn = connection number on the BIU to be set - an element of CN
ai = new outbound access interval - an element of AI or NULL

s-caii(b, cn, ai) - sets the current inbound access interval
b = BIU to be set - an element of B
cn = connection number on the BIU to be set - an element of CN
ai = new inbound access interval - an element of AI or NULL

NULL values indicate that the connection is being terminated.

Let there be a function, set-state-info(), that is to be used as a convenient way to set and reset the state information described above. It is defined in terms of the last four functions defined above and is executed exclusively at the request of the AC.

Set-state-info(b1, cn1, b2, cn2, t, ail, ai2) -> true if and only if
s-ct(b1, cn, t), s-cb(b1, cn1, b2),
s-ccn(b1, cn1, cn2), s-caio(b1, cn, ail), and
s-caii(b1, cn, ai2)

and if

t = NULL, b2 = NULL, cn2 = NULL, ail = NULL, or ai2 = NULL then t = NULL, b2 = NULL, cn2 = NULL, ail = NULL, and ai2 = NULL.

The second condition exists so that all of the status information is reset whenever any part of the status information is reset.

The AC must keep track of all open connections. When a connection is opened, the AC records the event by entering all of the pertinent information in the connection table (CT). The CT is defined as a subset of Connection-space which is the Cartesian Product:

Connection-space=(B X CN X B X CN X T X AI X AI).

The AC uses the function add-connection to record the opening of a connection in the CT. This function makes two entries into the table, one for each BIU involved. Both entries contain the same information but rearranged so that each BIU's status information is reflected. The definition is as follows:

add-connection(b1, cn1, b2, cn2, t, ail, ai2) returns true if
(b1, cn1, b2, cn2, t, ail, ai2) and
(b2, cn2, b1, cn1, t, ai2, ail)

have been added to the CT. The reason that the access intervals are reversed in the two tuples is that if two BIU's are communicating; one's outgoing traffic will be the other's incoming.

The AC uses the function del-connection to delete entries in the CT. Unlike add-connection, this function only effects one entry in the CT. When a BIU notifies the AC that it is through with a connection, the AC calls this function to remove that BIU's entry. This function must be called twice to completely close a connection. del-connection(b1, cn1) returns true if a tuple in the form of (b1, cn1, b, cn, t, ail, ai2) is removed from the CT, where b, cn, t, ail, ai2 need not be specified. Since the ordered pair (b1, cn1) uniquely determines one of b1's connections, it is unnecessary to completely specify the tuple.

It is now possible to define send-packet() and receive-packet(). Both of these functions return true only when their respective tasks have successfully been completed. Each takes five arguments defined as follows:

sb = source BIU - an element of B
scn = source BIU's connection number - an element of CN
db = destination BIU - an element of B
dcn = destination BIU's connection number - an element of CN
packet = the entire packet being sent or received - an element of P

It is implicit in the definition of both functions that sb, scn, db, and dcn properly reflect the source and destination address of the packet. They are passed as separate

arguments for easier reference and understanding.

Definition of send-packet(sb, scn, db, dcn, packet):

```
If [
  [mode(sb) = 0 and
   cb(sb, scn) = db and
   ccn(sb, scn) = dcn and
   dominate(label(packet), min(caio(sb, scn)))
   and dominate(max(caio(sb, scn)),
   label(packet))]
or [mode(sb) = 1 and
   cb(sb, scn) = db and
   ccn(sb, scn) = dcn and
   s-label(packet, max(caio(sb)))]]
or db = 'B-AC'
or sb = 'B-AC'
]
Then send-packet(sb, scn, db, dcn, packet) -> True
Else send-packet(sb, scn, db, dcn, packet) -> False
```

Definition of receive-packet(sb, scn, db, dcn, packet):

```
If [ id() = dh and
  unaltered(packet) and
  [ cb(db, dcn) = sb and
    ccn(db, dcn) = scn and
    dominate(label(packet), min(caii(db, dcn))) and
    dominate(max(caii(db, dcn)), label(packet))]
  or
  sb = 'B-AC'
  or
  db = 'B-AC']],
then receive-packet(sb, scn, db, dcn, packet) -> True
else receive-packet(sb, scn, db, dcn, packet) -> False
```

where 'B-AC' is the AC's BIU and unaltered is the function which returns true if and only if the packet arrived unaltered.

Finally, it is possible to define connect-open and connect-close. Each returns true when a connection has actually been opened or closed. Both functions are defined recursively in terms of each other.

When opening a connection between two BIU's, what actually happens depends on which BIU's are involved. When the B-AC is the requesting BIU, it generates a new connection number and informs the other BIU that a connection is being opened. The other BIU generates its own new connection number, sets its state information, and transmits its connection number in the process of notifying the B-AC that it is ready. The B-AC now has the necessary information to set its own state information. When finished, the B-AC notifies the AC that the connection has been opened so that the AC may add it to the connection table. Any BIU wishing to open a connection with the B-AC sends an open request to the B-AC, and the B-AC then proceeds as if it initiated the open request, following the steps given above.

No BIU can go directly to another BIU to request an open connection. The AC, through the B-AC, must be consulted for all such requests. The requesting BIU (b1) must open a connection with the B-AC to ask the AC for permission to open a connection to another BIU (b2) (for which b1 has already assigned a connection number). If the AC denies the request, then B-AC closes the connection. If the AC approves the request, then B-AC opens a separate connection with b2 who is informed of the b1 request. If b2 rejects the request, the B-AC notifies b1 and both connections are closed. If the request is accepted, however, b2 is instructed to generate a connection number, set its status information, and report back to the B-AC. B-AC sends the b2 connection number to b1 with instructions to set its status information and report back to the B-AC. Since they now consider the connection between them open, b1 and b2 both close their connection with the B-AC, and

the AC adds the connection to the CT.

In closing a connection, the action taken also depends on which BIU's are involved. A BIU considers a connection closed when its half is closed. The B-AC closes its connections by resetting its status information and notifying the AC to delete the connection from the CT. It has been assumed that the other BIU would initiate the close of all connections involving the B-AC.

A BIU closing a connection with the B-AC must notify the B-AC so that instructions may be issued to reset the BIU's status. When confirmation has arrived that the other BIU has been reset, the B-AC resets its status information and notifies the AC to delete the connection from the CT. A BIU closing connections that do not involve the B-AC must open a connection with the B-AC to notify the AC that the connection is closing, reset its status information when instructed to do so, and close the connection with the B-AC. The AC deletes each half of the connection as it is closed.

The following constants are used in connect-open and connect-close:

B-AC = The AC's BIU.

C = A type of connection used for control information.

Control-AI = The access interval used in a connection of type C.

SET-STATE-INFO = Packet instructing a BIU to set its state information. The state information is contained in the packet.

OPEN-REQ = Packet requesting the opening of a connection. The necessary information is contained in the packet.

OPEN-ACK = Packet notifying receiver that the proposed connection has been accepted.

OPEN-NAK = Packet notifying receiver that the proposed connection has been refused.

OPENED = Packet notifying receiver that a connection has been opened.

CLOSED = Packet notifying receiver that current connection is closing.

The parameters of connect-open are:

b1 = biu requesting connect-open - element of B.

cn1 = parameter in which connection number for b1 is to be returned - element of CN.

b2 = biu connect-open requested of - element of B.

cn2 = parameter in which connection number for b2 is to be returned - element of CN.

t = type of connection - element of T.

aio = outbound AI for b1 (inbound for b2) - element of AI

aii = inbound AI for b1 (outbound for b2) - element of AI

The definition of connect-open follows:

connect-open(b1, cn1, b2, cn2, t, aio, aii)

/* Connection from the B-AC to any BIU */

IF (b1 = 'B-AC')

THEN

cn1 = new-cn(b1)

send-packet(b1, cn1, b2, cn2, 'SET-STATE-INFO')

receive-packet(b1, cn1, b2, cn2, 'SET-STATE-INFO')

cn2 = new-cn(b2)

set-state-info(b2, cn2, b1, cn1, 'C',

'CONTROL-AI', 'CONTROL-AI')

```

send-packet(b2, cn2, b1, cn1, 'OPENED')
receive-packet(b2, cn2, b1, cn1, 'OPENED')
set-state-info(b1, cn1, b2, cn2, 'C',
'CONTROL-AI', 'CONTROL-AI')
add-connection(b1, cn1, b2, cn2, 'C',
'CONTROL-AI', 'CONTROL-AI')

```

```

/* Connection from any BIU to the B-AC */
ELSE IF (b2 = 'B-AC')

```

```

THEN
send-packet(b1, 'NULL', b2, 'NULL', 'OPEN-REQ')
receive-packet(b1, 'NULL', b2, 'NULL', 'OPEN-REQ')
connect-open(b2, b1, 'C', 'CONTROL-AI', 'CONTROL-AI')

```

```

ELSE /* Connection for any other two BIU's */

```

```

connect-open(b1, cn3, 'B-AC', cn4, 'C',
'CONTROL-AI', 'CONTROL-AI')
send-packet(b1, cn3, 'B-AC', cn4, 'OPEN-REQ')
receive-packet(b1, cn3, 'B-AC', cn4, 'OPEN-REQ')
IF (open-ok(b1, b2, t, ail, ai2) )
THEN
connect-open('B-AC', cn5, b2, cn6, 'C',
'CONTROL-AI', 'CONTROL-AI')
send-packet('B-AC', cn5, b2, cn6, 'OPEN-REQ')
receive-packet('B-AC', cn5, b2, cn6, 'OPEN-REQ')
send-packet(b2, cn6, 'B-AC', cn5, RESPONSE)
receive-packet(b2, cn6, 'B-AC', cn5, RESPONSE)
IF (RESPONSE = 'OPEN-ACK')

```

```

THEN
cn1 = new-cn(b1); cn2 = new-cn(b2)
send-packet('B-AC', cn5, b2, cn6, 'SET-STATE-INFO')
receive-packet('B-AC', cn5, b2, cn6,
'SET-STATE-INFO')
set-state-info(b2, cn2, b1, cn1, t, aii, aio)
send-packet(b2, cn6, 'B-AC', cn5, 'OPENED')
receive-packet(b2, cn6, 'B-AC', cn5, 'OPENED')
send-packet('B-AC', cn4, b1, cn3, 'SET-STATE-INFO')
receive-packet('B-AC', cn4, b1, cn3,
'SET-STATE-INFO')
set-state-info(b1, cn1, b2, cn2, t, aio, aii)
send-packet(b1, cn3, 'B-AC', cn4, 'OPENED')
receive-packet(b1, cn3, 'B-AC', cn4, 'OPENED')
add-connection(b1, cn1, b2, cn2, t, aio, aii)
connect-close(b1, cn3, 'B-AC', cn4)
connect-close(b2, cn6, 'B-AC', cn5)

```

```

ELSE
connect-close(b2, cn6, 'B-AC', cn5)
send-packet('B-AC', cn4, b1, cn3, 'OPEN-NAK')
receive-packet('B-AC', cn4, b1, cn3, 'OPEN-NAK')
connect-close(b1, cn3, 'B-AC', cn4)

```

```

ELSE
send-packet('B-AC', cn4, b1, cn3, 'OPEN-NAK')
receive-packet('B-AC', cn4, b1, cn3, 'OPEN-NAK')
connect-close(b1, cn3, 'B-AC', cn4)

```

The parameters of connect-close are:

```

b1 = BIU wishing to close its portion of a connection
cn1 = b1's connection number to be closed
b2 = Other BIU involved in connection
cn2 = b2's connection number

```

The definition of connect-close follows:

```

connect-close(b1, cn1, b2, cn2)

```

```

IF (b1 = 'B-AC')
THEN
set-state-info(b1, cn1, 'NULL', 'NULL', 'NULL',
'NULL', 'NULL')
del-connection(b1, cn1)

```

```

ELSE IF (b2 = 'B-AC')
THEN
send-packet(b1, cn1, b2, cn2, 'CLOSED')
receive-packet(b1, cn1, b2, cn2, 'CLOSED')
send-packet(b2, cn2, b1, cn1, 'SET-STATE-INFO')
receive-packet(b2, cn2, b1, cn1, 'SET-STATE-INFO')
send-packet(b1, cn1, b2, cn2, 'CLOSED')
receive-packet(b1, cn1, b2, cn2, 'CLOSED')

```

```

set-state-info(b1, cn1, 'NULL', 'NULL', 'NULL',
'NULL', 'NULL')
del-connection(b1, cn1)
connect-close(b2, cn2, b1, cn1)

```

ELSE

```

send-packet(b1, cn1, b2, cn2, 'CLOSED')
receive-packet(b1, cn1, b2, cn2, 'CLOSED')
connect-open(b1, cn3, 'B-AC', cn4, 'C',
'CONTROL-AI', 'CONTROL-AI')
connect-open(b2, cn5, 'B-AC', cn6, 'C',
'CONTROL-AI', 'CONTROL-AI')
send-packet(b1, cn3, 'B-AC', cn4, 'CLOSED')
send-packet(b2, cn5, 'B-AC', cn6, 'CLOSED')
receive-packet(b1, cn3, 'B-AC', cn4, 'CLOSED')
receive-packet(b2, cn5, 'B-AC', cn6, 'CLOSED')
send-packet('B-AC', cn4, b1, cn3, 'SET-STATE-INFO')
send-packet('B-AC', cn6, b2, cn5, 'SET-STATE-INFO')
receive-packet('B-AC', cn4, b1, cn3, 'SET-STATE-INFO')
receive-packet('B-AC', cn6, b2, cn5, 'SET-STATE-INFO')
set-state-info(b1, cn1, 'NULL', 'NULL', 'NULL',
'NULL', 'NULL')
set-state-info(b2, cn2, 'NULL', 'NULL', 'NULL',
'NULL', 'NULL')
send-packet(b1, cn3, 'B-AC', cn4, 'CLOSED')
send-packet(b2, cn5, 'B-AC', cn6, 'CLOSED')
receive-packet(b1, cn3, 'B-AC', cn4, 'CLOSED')
receive-packet(b2, cn5, 'B-AC', cn6, 'CLOSED')
del-connection(b1, cn1)
del-connection(b2, cn2)
connect-close(b1, cn3, 'B-AC', cn4)
connect-close(b2, cn6, 'B-AC', cn6)

```

Let there be a set, E, which is the set of all possible events that occur on the network. Some elements of E would be things such as opening a connection, delivering a packet, or a new host added to the LAN. Some of these events are security related and, as such, need to be audited. These might occur at the BIU (an improperly labeled packet arrives at the BIU) or at the AC (a connection request is denied).

Let there exist a function, security-relevant(e), where e is an element of E that determines if an event is security relevant. Let there also be a function, audit(e), that causes the time, place, and involved parties of that event to be logged in audit files located on the AC.

It is now possible to present mathematical conditions that must hold true if the five security rules are being enforced. Strictly speaking, Rule 4 is unnecessary because it can be implied from Rule 3. It has been included to emphasize the importance of packets arriving at their destination unaltered. The five security rules are as follows:

Rule 1: For all p, an element of P, there exists an l, an element of C-space, such that label(p) = l.

Rule 2: For any b, an element of B; cn, an element of CN; and p, an element of P; b may transmit p on cn if and only if send-packet(b, cn, cb(b, cn), ccn(b, cn), p) -> True.

Rule 3: For any b, an element of B; cn, an element of CN; and p, an element of P; b may deliver a p received on cn if and only if receive-packet(cb(b, cn), ccn(b, cn), b, cn, p) -> True.

Rule 4: For any b1 and b2, elements of B; cn1 and cn2, elements of CN; and p, an element of P; if receive-packet(b1, cn1, b2, cn2, p) -> True, then unaltered(p) -> True.

Rule 5: For all e, elements of E, if security-relevant(e) -> True, then audit(e) -> True.

B. The LAN Model

In modeling the secure operation of the LAN, the secure operation of the BIU and the AC needs to be described. Each will be described as a collection of state machines. The states and the events that cause transitions between them will be described. The communications between these devices will then be modeled to complete the description of the total operation of the LAN. At that point, there should be a model from which the security-relevant points can be proven.

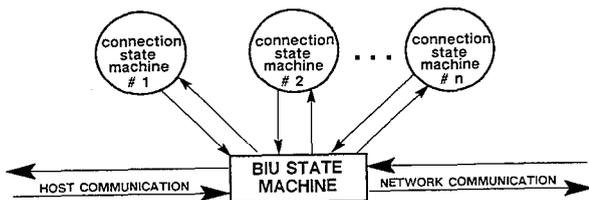
1. The BIU

The first device to be described is the BIU. The main function of the BIU is to send and receive packets for the attached host. To do this, the BIU must be able to establish, maintain, and close connections. It must be able to distinguish to which connection a packet belongs as well as whether that packet is permitted to be sent or received. The BIU must be able to do this for any number of connections, limited only by its own physical resources of those of the attached host. It must be able to communicate with the AC through the B-AC and respond to NSO commands issued through the AC. Finally, the BIU must realize when a security-relevant event has occurred and record that event in an audit log.

The behavior of a BIU is modeled by describing the life of each connection, from birth until death, that a BIU manages with a separate state machine. Each of these state machines, called Connection State Machines (CSM's), models those functions in the BIU that establish, maintain, and then close a particular connection. Included in this functionality is all of the security checking that is done for that connection.

These CSM's, however, do not model the entire operation of the BIU. Some functionalities not modeled are the BIU's ability to communicate with the network or attached hosts and its audit capability. A separate state machine, called the BIU State Machine (BSM), does this for each of the CSM's. The BSM takes the input to the BIU and decides to which connection it belongs and then passes it to the CSM handling that connection. If a CSM is not currently active to handle the input, the BSM initiates one that can.

The real purpose of the BSM is to model the physical operations of the BIU. When input comes into the BIU, the BSM checks that input, sends it to a CSM for a decision on what action to take, waits for a response, and takes action appropriate to that response (see Figure 2).



a. The states for the BSM are as follows:

(1) wait - The BSM waits for input to the BIU from the network, the attached host, or one of the CSM's.

(2) check-external-input - The BIU examines input from the network or the host. It rejects the input if it is not addressed for the BIU or has been damaged in some way. It is here that the BIU, if operating in mode one, inserts security labels into the packet headers.

(3) pass-to-CSM - The input is passed to the appropriate CSM. If there is no CSM available to handle the input, a state is entered that will spawn one.

(4) spawn-CSM - A new instantiation of a CSM is created to which the input is passed.

(5) audit - Security-relevant events are logged in the BIU.

(6) deliver - Packets are delivered to the attached host.

(7) send - Packets are transmitted out on the bus.

(8) change-mode - The operating mode of the BIU is changed.

(9) report-failure - An attempt is made to report any BIU-detected failure to the AC. This attempt may or may not succeed depending on the nature of the failure.

(10) disconnect - The BIU is electrically disconnected from the bus. This should happen after any failure is detected regardless of whether or not it has been successfully reported to the AC.

b. Most of the events that cause BSM state transitions are the result of some output from one of the CSM's. The events are as follows:

(1) external-input - A packet has arrived from the host or network.

(2) new-CSM - The packet that just arrived requires a new CSM to handle it.

(3) audit-req - A CSM has signaled that some event needs auditing.

(4) audit-full - The event just audited caused the audit files to be larger than some threshold value.

(5) deliver-req - A CSM has signaled that a packet is ready to be delivered to the host.

(6) send-req - A CSM has signaled that a packet is ready to be sent out on the network.

(7) change-mode-req - A CSM has signaled that the AC has requested a mode change for the BIU.

(8) disconnect-req - A CSM has signaled that the AC has requested that the BIU electrically disconnect itself from the bus.

(9) done - The action of the current state has been successfully completed.

(10) reject - The packet that just arrived from the host or network was damaged or not intended for the BIU.

(11) failure - The action of the current state has not been successfully completed, or the BIU has detected some hardware failure. This may happen in any state.

(12) reset - The BIU has just been connected to the network and begun operation.

(See Figure 3 for a BSM state diagram and Figure 4 for a BSM state transition table.)

c. There are three possible ways for a BIU to be involved in a connection. It can be attached to a host that is initiating a connection, attached to a host that is the recipient of a connection, or attached to the AC. The CSM manages individual connections and must, therefore, be able to handle all three cases. As a result, the CSM is considerably more complicated than the BSM. Its states are as follows:

(1) birth - This is the initial state of the CSM. What role the BIU is to play in the connection

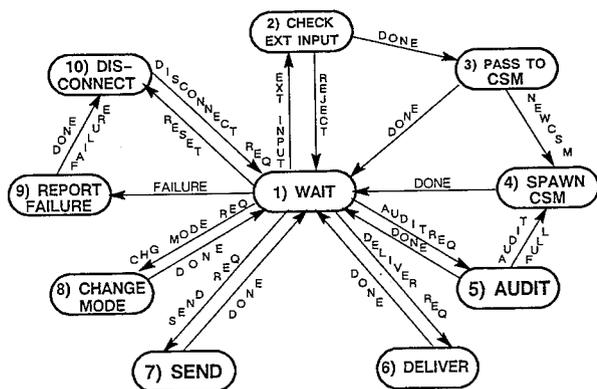


FIGURE 3: BSM STATE DIAGRAM

STATE	EVENT	NEW CSM	AUDIT FULL	AUDIT REQ	DELIVER REQ	SEND REQ	CHG MODE REQ	DISCONNECT REQ	DONE	REJECT	RESET	FAILURE
WAIT	1	2	5	6	7	8	10			3	1	9
CHECK EXTERNAL INPUT	2											
PASS TO CSM	3	4								1	9	
SPAWN CSM	4									1	9	
AUDIT	5		4							1	9	
DELIVER	6									1	9	
SEND	7									1	9	
CHANGE MODE	8									1	9	
REPORT FAILURE	9									10	10	
DISCONNECT	10											1

FIGURE 4: BSM STATE TRANSITION TABLE

determines the next state of the CSM.

(2) send-open-req - The BIU just received a connection open request from its attached host. The BSM is signaled to forward this request to the AC for approval.

(3) deliver-open-req - The BIU just received a connection open request that the AC has approved. The BSM is signaled to deliver this request to the attached host.

(4) wait-open - Either the originating BIU is waiting for a response from the AC, or the receiving BIU is waiting for a response from the attached host.

(5) notify-AC-open-ack - The receiving BIU has just received approval from the attached host that the connection may be opened. The BSM is signaled to notify the AC.

(6) notify-host-open-ack - The originating BIU has just received approval from the AC that the requested connection may be opened. The BSM is signaled to notify the attached host.

(7) notify-AC-open-nak - The receiving BIU has just received word that its attached host has refused the proposed connection. The BSM is signaled to notify the AC.

(8) notify-host-open-nak - The originating BIU has just received word that the proposed connection, for some unknown reason, has been refused. The BSM is signaled to notify its attached host.

(9) wait-set-status-req - A host-to-host connection is about to be opened or closed. The CSM is waiting for the AC to instruct it to set or reset the connection status information.

(10) set-status - The AC has instructed that the connection status be set, and this is done.

(11) notify-AC-status-set - The connection status of the CSM has been successfully set. The BSM is signaled to notify the AC.

(12) audit - One of four auditable events has just occurred in the CSM: a packet has arrived that cannot be delivered, a packet has arrived that cannot be sent, a connection has been opened, or one has been closed. All four events must be audited, and the BSM is signaled to do so.

(13) wait-input - A connection is presently in progress. The CSM is waiting from input from the network or the host.

(14) check-net-send - A packet has arrived at the BIU from the attached host. The packet is checked with respect to security to determine if it may be sent out in this connection.

(15) send - A packet has been determined fit to send in this connection. The BSM is signaled to send it out on the network.

(16) check-host-delivery - A packet has arrived at the BIU from the network. In this state, the packet is checked with respect to security to determine if it may be delivered in this connection. The packet is also checked to see if it is a close connection request.

(17) deliver - A packet has been determined fit to deliver in this connection. The BSM is signaled to deliver it to the attached host.

(18) notify-host-closed - The BIU has received notice, either from a host or the AC, that the current connection has been closed. The BSM is signaled to notify the attached host.

(19) notify-AC-closed - The current connection has been closed. The AC is notified so that the BIU status may be reset.

(20) notify-host-not-authorized - The host has attempted to send unauthorized information out on the network. The BSM is signaled to notify the host of the error.

(21) send-audit - The BIU has received a request from the AC to begin sending its locally-stored audit data. The BSM is signaled to send a packet of audit data.

(22) wait-ok-to-send-audit - The BIU is ready to send the AC audit data. The CSM is waiting for confirmation that the AC is ready to receive it.

(23) notify-AC-audit-full - The BSM has realized that its audit files are nearly full. The CSM signals the BSM to notify the AC.

(24) notify-AC-audit-sent - The audit files have been completely sent to the AC. The BSM is signaled to notify the AC.

(25) chmod - The BIU has received a request from the AC to change its operating mode. The BSM is signaled to do so.

(26) notify-AC-mode-changed - The operating mode of the BIU has just been changed. The BSM is signaled to notify the AC.

(27) disconnect - A disconnect request has arrived at the BIU from the AC. The BSM is signaled to electrically disconnect the BIU from the network.

(28) death - The connection has been completely closed and the CSM is no longer needed. The CSM is terminated.

d. The events that cause CSM state transitions are as follows:

(1) host-AC-open - The BIU has received a message from its host requesting a connection to be opened. This is one of the entry events to the CSM.

(2) AC-host-open - The BIU has received a packet from the AC informing it that another host wishes to open a connection to it. This is one of the entry events to the CSM.

(3) host-AC-open-ack - The BIU has been notified that its host has accepted the proposed connection.

(4) host-AC-open-nak - The BIU has been notified that its host has not accepted the proposed connection.

(5) AC-host-open-ack - The BIU has been notified that its host's connection request has been approved and will be opened.

(6) AC-host-open-nak - The BIU has been notified that its host's connection request has not been approved and will not be opened.

(7) set-stat-req - A packet has arrived at the BIU from the AC requesting that the BIU's connection status information be changed.

(8) from-host - While involved in a connection, the BIU has received a message from its host addressed to that connection.

(9) from-net - While involved in a connection, the BIU has received a packet from the network addressed to that connection.

(10) not-authorized - Either a packet from the network or a message from the host arrived at the BIU and cannot be passed through it.

(11) closed - Three things can cause this event: a host notifies its BIU that the current connection is over, a packet arrives at the BIU signifying the end of the connection, or the closing of the connection was just audited by the BIU.

(12) kill-con-req - A packet from the AC just arrived at the BIU instructing it to close that connection.

(13) BIU-AC-start - A packet just arrived at the B-AC from some other BIU requesting a dialogue with the AC, usually regarding the opening or closing of a connection. This is one of the entry events to the CSM.

(14) AC-BIU-start - A message just arrived at the B-AC from the AC initiating a dialogue with some other BIU, usually regarding the opening or closing of a connection. This is one of the entry events to the CSM.

(15) BIU-AC-end - The BIU-to-AC communication has ended.

(16) AC-BIU-end - The AC-to-BIU communication has ended.

(17) done - The action in the current

state has been successfully completed.

(18) audit-full - The BSM has realized that the audit files are nearing capacity and is requesting that they be sent to the AC. This is one of the entry events to the CSM.

(19) audit-req - A packet has arrived at the BIU from the AC requesting the BIU's audit information be sent to the AC. This is one of the entry events to the CSM.

(20) more-audit - Audit information has been sent from the BIU to the AC, but there is still more to send.

(21) chmod-req - A packet has arrived at the BIU from the AC requesting that BIU to change its operating mode. This is one of the entry events to the CSM.

(22) disconnect-req - A packet has arrived at the BIU from the AC instructing it to disconnect itself from the bus. This is one of the entry events to the CSM.

(See Figure 5 for a CSM state diagram and Figure 6 for a CSM state transition table.)

NOTE: There is a significant deficiency in the CSM as it has been described. It has been assumed that a connection already exists when communications occur between the B-AC and another BIU, but in reality, a connection would actually have to be opened and then closed for such a communication to take place. This is, however, reflected in the definitions of connect-open and connect-close and does not cause any of the total LAN functionality to be lost.

2. The Access Controller

The second device to be described is the AC. The AC has the primary responsibility to ensure the secure operation of the LAN. Since the BIU's turn to the AC for decisions regarding the permissibility of host-to-host connections, the AC must be capable of making those decisions. The AC must, therefore, know what is happening on the LAN at all times.

The AC maintains the MAC, DAC, and connection tables and is responsible for setting and resetting the connection status in the BIU's. It handles the LAN audit files. The AC is also the machine through which the NSO issues commands such as instructing a BIU to change operating modes, send audit data, break connections, and actually disconnect itself from the bus.

The AC is modeled similar to the BIU. There will be a state machine, called the Access Controller State Machine (ACSM), that describes the actual operations of the AC (Figure 7). These generic operations (such as sending and receiving data from the network, auditing, and accessing tables) are described independent of any particular network connection or NSO command.

There will also be instantiations of another state machine, the Connection State Machine - Access Controller (CSM-AC), to manage specific individual communications with a BIU or the NSO. These CSM-AC's signal the ACSM when they need some action performed.

a. The states for the ACSM are as follows:

(1) wait - The AC waits for input from the network, the NSO, or one of the CSM-AC's.

(2) pass-to-CSM-AC - Input is checked for proper format. The data is then passed to the appropriate CSM-AC. If there is no CSM-AC available to handle the input, a state is entered that will spawn one.

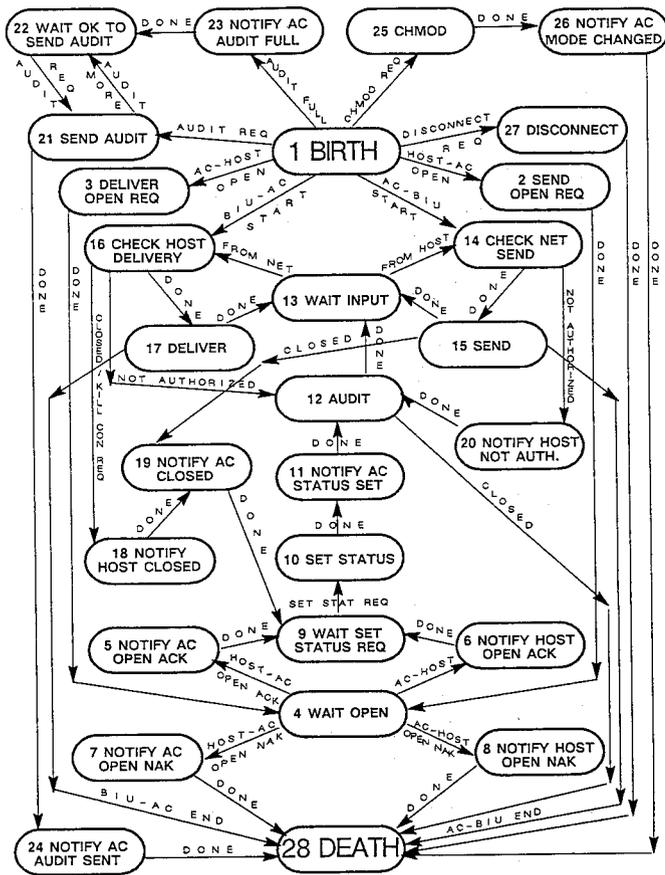


FIGURE 5: CSM STATE DIAGRAM

STATE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
BIRTH	1	2	3												16	14			23	21		25	27					
SEND OPEN REQ		2																4										
DELIVER OPEN REQ		3																4										
WAIT OPEN		4		5	7	6	8																					
NOT. AC OPEN ACK		5																9										
NOT. HOST OPEN ACK		6																9										
NOT. AC OPEN NAK		7																28										
NOT. HOST OPEN NAK		8																28										
WAIT SET STATUS REQ		9					10																					
SET STATUS		10																	11									
NOT. AC STATUS SET		11																	12									
AUDIT		12										28							13									
WAIT INPUT		13									14	16																
CHECK NET SEND		14										20																
SEND		15										19						28	13									
CHECK HOST DELIVERY		16									12	18	18					17										
DELIVER		17													28			13										
NOT. HOST CLOSED		18																19										
NOT. AC CLOSED		19																9										
NOT. HOST NOT AUTH.		20																12										
SEND AUDIT		21																24			22							
WAIT OK SEND AUDIT		22																		21								
NOT. AC AUDIT FULL		23																										
NOT. AC AUDIT SENT		24																										
CHMOD		25																										
NOT. AC MODE CHG		26																										
DISCONNECT		27																										
DEATH		28																										

FIGURE 6: CSM STATE TRANSITION TABLE

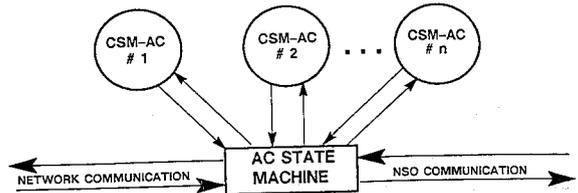


FIGURE 7: ACSM COMMUNICATIONS

(3) spawn-CSM-AC - A new instantiation of a CSM-AC is created, and the input is passed to it.

(4) audit - Security-relevant events occurring in the AC are logged in audit files. This includes entering audit data received from individual BIU's.

(5) access-tables - Either the MAC, DAC, or connection tables are read from or written to.

(6) net-output - Data is sent to the B-AC for transmission.

(7) NSO-output - The NSO is notified of some event, either the completion of one of his requests or some problem with the LAN.

(8) disconnect - The AC is no longer in control of the network. Either the LAN continues operating as it was when the state was entered with no further opens or closes, or an alternate AC is notified to assume the AC's responsibilities in the case of a redundant AC. In the second case, the assumption has been made that the two, or maybe more, AC's have been kept identical.

b. The events that cause ACSM state transitions are as follows:

(1) input - A message has just arrived at the AC from either the AC or the NSO.

(2) new-CSM-AC - There is no CSM-AC currently expecting the last received message; therefore, a state must be entered to spawn one.

(3) access-req - A CSM-AC has signaled a requirement to access one of the tables kept by the AC.

(4) audit-req - A CSM-AC has signaled that some event needs to be audited or that a BIU's audit information needs to be saved.

(5) audit-full - The last event audited caused the audit files to be larger than some threshold value.

(6) net-output-req - A CSM-AC has signaled that some message needs to be sent to a BIU.

(7) NSO-output-req - A CSM-AC has signaled that some message needs to be sent to the NSO.

(8) done - The action of the current state has been successfully completed.

(9) reject - The input to the ACSM was unreadable.

(10) AC-disconnect - The AC has instructed the B-AC to disconnect itself from the network.

(11) failure - A major failure has occurred in the AC.

(12) reset - The AC has just been activated and is assuming control of the LAN.

(See Figure 8 for an ACSM state diagram and Figure 9 for an ACSM state transition table.)

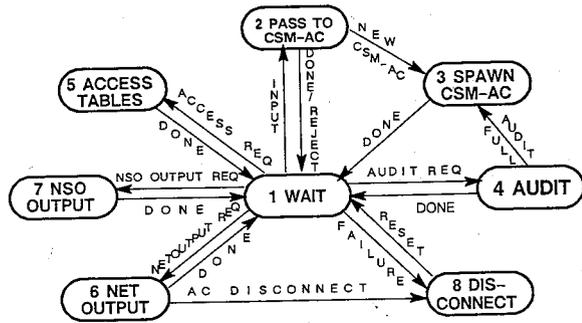


FIGURE 8: ACSM STATE DIAGRAM

STATE	INPUT	NEW CSM-AC	ACCESS REQ	AUDIT FULL	NET OUTPUT REQ	NSO OUTPUT REQ	DONE	REJECT	AC DISCONNECT	FAILURE	RESET
WAIT	1	2	5	4	6	7				8	
PASS TO CSM-AC	2	3					1	1		8	
SPAWN CSM-AC	3						1			8	
AUDIT	4			3			1			8	
ACCESS TABLES	5						1			8	
NET OUTPUT	6						1		8	8	
NSO OUTPUT	7						1			8	
DISCONNECT	8									8	1

FIGURE 9: ACSM STATE TRANSITION TABLE

c. The states for the CSM-AC are as follows:

(1) birth - This is the initial state of the CSM-AC. What action the AC takes on entry into the CSM-AC is determined by the reason it was spawned.

(2) get-MAC/DAC-info - The AC needs to make a decision about a connection open request and must consider the pertinent MAC and DAC information. In this state, the ACSM is signaled to retrieve this information.

(3) check-open-req - The open request just received is checked against the MAC and DAC information just retrieved.

(4) forward-open req - The open request has been approved by the AC, and the ACSM is signaled to forward the request to the recipient host.

(5) wait-open - The AC is waiting for a response to the forwarded open request from the recipient host.

(6) notify-host-open-nak - The open request in question has been denied either by the AC or by the recipient host, and the ACSM is signaled to notify the requesting host.

(7) send-status-info - A connection between two hosts is about to be opened or closed, and the BIU's need to be instructed to change their connection status information. The ACSM is signaled to instruct each one of the BIU's to change this information. This state will be entered twice during a connection opening; the first to set the recipient host's BIU and the second for the requesting host's BIU. During a close, each BIU will notify the AC it

is closing. Therefore, two separate CSM-AC's will be spawned, each entering this state only once.

(8) wait-status-set - After instructing the BIU to change its status information, the CSM-AC waits for confirmation that the information has been changed.

(9) notify-host-open-ack - The AC and the recipient host have approved a host's open request, and the status information in the recipient host's BIU has also been set for the connection. The ACSM is signaled to notify the requesting host of this and to expect its status to be set for the connection.

(10) update-connection-table - A connection has just been opened or closed, and the AC's connection tables are updated to reflect this.

(11) audit - The action performed during a CSM-AC's life must be audited before its death. The ACSM is signaled to record some auditable event in the AC's audit files.

(12) notify-BIU-send-audit - A condition has arisen, either a BIU message or an NSO request, that requires a BIU to send its audit data to the AC. The ACSM is signaled to notify the BIU that the AC is ready to receive the audit data.

(13) wait-audit-data - The CSM-AC is waiting for a BIU to transmit some audit data.

(14) store-audit-data - The ACSM is signaled to store a BIU'S audit data in the AC'S audit files.

(15) notify-BIU-change-mode - A request has come from the NSO to change the operating mode of a BIU. The ACSM is signaled to instruct the BIU to change its operating mode.

(16) wait-mode-changed - The CSM-AC is waiting for confirmation that the BIU changed its operating mode.

(17) notify-BIU-disconnected - A request has come from the NSO to disconnect a BIU from the LAN. The ACSM is signaled to instruct the BIU to disconnect itself.

(18) notify-NSO-BIU-disconnected - The ACSM is signaled to notify the NSO that the BIU has been instructed to disconnect itself from the LAN. There is no actual confirmation from the LAN that the disconnect actually happened; therefore, the NSO should probably verify the disconnect for himself.

(19) notify-NSO-AC-audit-full - The ACSM has noticed that the last piece of audit data caused the AC'S audit storage area to grow larger than some threshold value. The ACSM is signaled to notify the NSO of this fact.

(20) update-DAC-table - The NSO has instructed the AC that a host's DAC table is to be updated, and the ACSM is signaled to make the changes.

(21) update-MAC-table - The NSO has instructed the AC that a host's MAC table is to be updated, and the ACSM is signaled to make the changes.

(22) notify-NSO-table updated - The ACSM is signaled to notify the NSO that the requested changes have been made.

(23) notify-BIU-connection-killed - The NSO has instructed that a connection be terminated. The ACSM is signaled to notify one of the BIU's to kill the connection at its end. To completely kill an active connection, two separate CSM-AC's need to be spawned - one for each BIU.

(24) wait-closed - The CSM-AC realizes that a connection was just killed and is waiting for a BIU to send the connection closed response to the AC.

(25) death - The task that the CSM-AC was spawned to perform has been completed and audited, and the life of the CSM-AC is terminated.

d. The events that cause CSM-AC state transitions are as follows:

(1) host-AC-open-req - A host is requesting permission to open a connection with another host.

(2) host-AC-open-ack - A host has just accepted a connection open request that had been forwarded by the B-AC.

(3) host-AC-open-nak - A host has just refused a connection open request that had been forwarded by the AC.

(4) not-authorized - A connection open request failed to pass the MAC and DAC checks.

(5) stat-set-#1 - The AC has received confirmation that the status information of a BIU was successfully set. This event occurs when the BIU in question was the recipient of the open request.

(6) stat-set-#2 - The AC has received confirmation that the status information of a BIU was successfully set. This event occurs when the BIU in question was the originator of an open request or is in the process of closing its end of a connection.

(7) closed - A BIU has notified the AC that its end of a connection has been closed. This could happen as the result of a normal close or from the request of the NSO.

(8) NSO-kill-connection - The NSO has requested that a BIU close its end of a connection.

(9) BIU-AC-audit-full - A BIU has notified the AC that its audit files are full and need to be sent to the AC.

(10) NSO-audit-req - The NSO has requested that a BIU send its audit files to the AC.

(11) audit-data - The input just received was BIU audit data.

(12) audit-sent - The BIU has informed the AC that all of its audit data has been sent.

(13) NSO-disconnect-BIU - The NSO has requested that a BIU be disconnected from the LAN.

(14) NSO-update-MAC-table - The NSO has requested that a change be made to a host's MAC tables.

(15) NSO-update-DAC-table - The NSO has requested that a change be made to a host's DAC tables.

(16) NSO-change-BIU-mode - The NSO has requested that a BIU's operating mode be changed.

(17) BIU-mode-changed - The AC has received confirmation that a BIU's operating mode has been changed.

(18) AC-audit-full - The ACSM has just signaled that the AC's audit data storage area is nearing capacity and that the NSO needs to be notified.

(19) done - The action in the current state has been successfully completed.

(See Figure 10 for a CSM-AC state diagram and Figure 11 for a CSM-AC state transition table.)

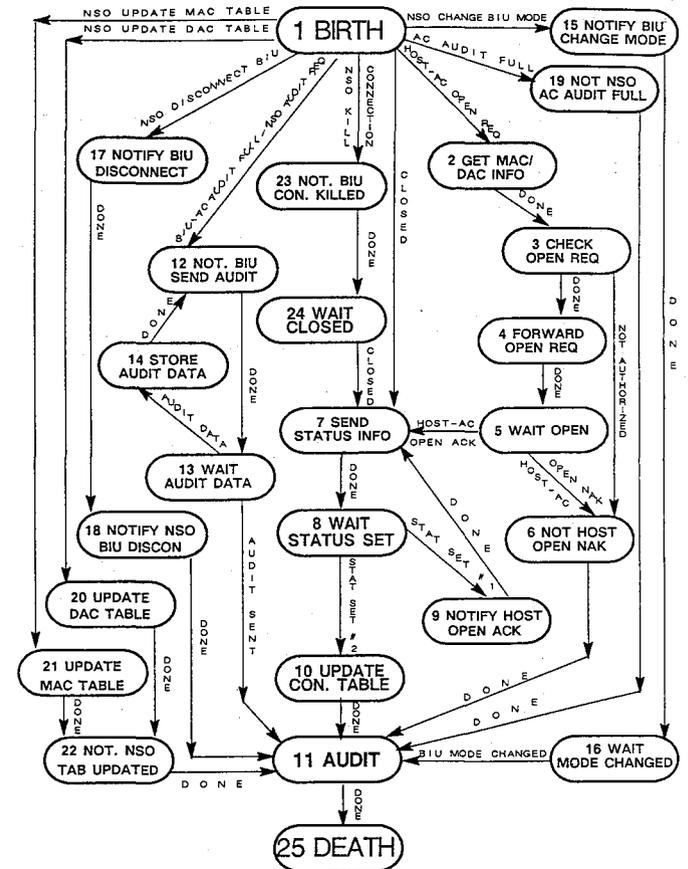


FIGURE 10 CSM-AC STATE MACHINE

EVENT	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
BIRTH	1	2					7	23	12	12				17	21	20	15		19	
GET MAC/DAC INFO																				3
CHECK OPEN REQ																				4
FORWARD OPEN REQ																				5
WAIT OPEN																				
NOTIFY HOST OPEN NAK																				11
SEND STATUS INFO																				8
WAIT STATUS SET																				
NOTIFY HOST OPEN ACK																				7
UPDATE CON. TABLE																				11
AUDIT																				25
NOTIFY BIU SEND AUDIT																				13
WAIT AUDIT DATA														14	11					
STORE AUDIT DATA																				12
NOT. BIU CHANGE MODE																				16
WAIT MODE CHANGED																				11
NOT. BIU DISCONNECTED																				18
NOT. NSO BIU DISCON.																				11
NOT. NSO AC AUDIT FULL																				11
UPDATE DAC TABLE																				22
UPDATE MAC TABLE																				22
NOT. NSO TABLE UPDATED																				11
NOT. BIU CON. KILLED																				24
WAIT CLOSED																				24
DEATH																				25

FIGURE 11: CSM-AC STATE TRANSITION TABLE

NOTE: The same deficiency exists in the description of the CSM-AC as that in the CSM. The CSM-AC does not account for the opening of connections with BIU's with which it communicates.

3. The Bus

The bus is the last part of the LAN that needs to be modeled. The bus is the cable that connects all of the BIU's on the LAN. When a BIU wishes to transmit a packet, it broadcasts it on the bus. If all goes well, it arrives at every BIU on the bus, including the sender. The destination BIU accepts the packet, and the rest ignore it. The bus never guarantees that the packet sent is the one received. That is the BIU's problem.

The bus is modeled with the assumption that there is a buffer at each interface to the bus. When a BIU wishes to send a packet, it writes that packet into the buffer at its bus interface. The property of the bus is to copy the contents, not necessarily correctly, to every other buffer. When something is written into one of those buffers, the BIU at that interface treats that as a received packet. It is up to that BIU to determine if that packet is correct or even addressed to it (see Figure 12).

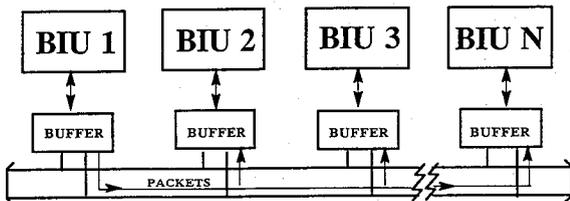


FIGURE 12: THE BUS

IV. SUMMARY

This paper is a first attempt at specifying and then modeling a security policy for an MLS LAN. It describes a policy that gives the reader an intuitive feeling of what it means for a LAN to operate securely by putting forth a list of rules that must be obeyed along with motivation for each rule. It then tries to formalize these rules. Finally, a fairly lengthy description of the LAN was presented by describing each device on the LAN with intercommunicating state machines.

Since this is a first attempt at the model, there are naturally some aspects of the LAN that have not been fully specified in the model. Also no formal proofs have been attempted. What is hoped, however, is that there is now a foundation which will evolve into a fully-specified MLS LAN model that is provably secure.

REFERENCES

- [1] DoD 5200.1R, The Department of Defense Information Security Program Regulation, July 1982.
- [2] *Department of Defense Trusted Computer Security Evaluation Criteria*, CSC-STD-001-83, 15 August 1983.
- [3] CDRL 145, "Formal Draft Subsystem Design Analysis Report - Engineering Report: LAN Interfaces." GTE Contract No. F19628-84-C-0052, 10 August 1982, Volume 4, Appendix C.

SECURITY IN OPEN SYSTEMS

A REPORT ON THE STANDARDS WORK OF ECMA'S TC32/TG9

T A Parker
ICL Defence Systems UK

ABSTRACT

TC32/TG9 is a recently formed Task Group within the European Computer Manufacturers Association standards body (ECMA). It has been tasked with defining an application-layer framework for Security in Open Systems, a framework which will ultimately lead to the definition of standard security support applications that communicate in the OSI environment using standard application-layer protocols.

This paper reports on some of the early work of TG9 completed mainly during 1986. It describes an informal secure systems model or framework, in which security is supported by a number of discrete security "facilities". The paper then goes on to report on some of the detailed work that has been started on analyzing requirements for the passing of security data around a distributed system. It addresses the topic of access authorization and offers a uniform approach which caters for a spectrum of access control schemes ranging from capability systems to access control lists.

ACKNOWLEDGEMENTS

The other major contributors to the work of TG9, of which this paper is only a part, have been J Kruys (NCS, Netherlands, and TG9 convenor), D Roberts (British Telecom, UK), N Pope (GEC, UK), D Pinkas (Bull, France), and A C Gale (ICL UK).

0. CONTENTS

1. INTRODUCTION
2. THE SECURE SYSTEMS MODEL
3. SECURITY FACILITIES

4. WALKTHROUGH
5. THE AUTHORIZATION MODEL
6. RELATIONSHIP TO THE DoD EVALUATION CRITERIA
7. CONCLUSION
8. REFERENCES

1. INTRODUCTION

TG9 is a Task Group of Technical Committee 32 (TC32) of the European Computer Manufacturers Association (ECMA). Its responsibility is to develop a framework for the provision of logical security in an Open Systems environment and to develop standards for security-related services and protocols, or protocol elements, as required for this environment.

The work of the TG9 group addresses the ISO application-layer view of a distributed system. It is aimed at developing standard security applications and standard communications protocols both between the applications themselves and between them and the productive applications with which they share the system. In some cases it is envisaged that standard protocol elements will be developed with which existing application-layer protocols can be extended.

The world of the TG9 framework is one of end users in control of entities that communicate via Application Service elements (Ref 1). The generic term application is used in the text that follows, to denote one of these entities; so in TG9 terms a file service, an

office mailbox, a print spooler, or a UNIX operating system offering general purpose computing facilities to its users are all examples of applications.

The TG9 group is primarily concerned with the ways in which applications interwork rather than how they are constructed internally. TG9 therefore concentrates its efforts on network-wide aspects of security, only looking inside applications in order to establish what externally communicated security data they may need (or at least benefit from) in order to do their own job. This split between views of security external to and internal to applications is fundamental to the approach and is further discussed in Section 2.

In either view, security is obtainable only via the implementation of a variety of control and monitoring functions, the requirements for which are determined according to whatever security policy is defined for the view. TG9's secure systems framework identifies a general set of these functions and divides them into elements, each one having a single coherent role to play in the provision of the total security picture.

These elements are referred to as Security Facilities and they are described in Section 3. The framework must also show how these elements interact with each other and consequently which combinations of elements are appropriate to form standard security support applications; this is a major area of current activity for TG9. A walkthrough of interactions that could occur when a user logs on to a system and attempts access to an application is given in Section 4. The walkthrough should be taken as

illustrative rather than definitive; it leads naturally into the second part of this paper (Section 5) which covers one major aspect of the detailed work being done within TG9. The topic is that of access authorization.

Whereas authentication relates to the process of proving claims of identity, authorization relates to the process of controlling access by already identified subjects to already identified protected objects*. The paper aims to show that existing ad hoc authorization methodologies can be fitted into a unifying framework in which apparently quite different techniques appear as different parts of a continuously varying spectrum. In particular the two authorization approaches characterized by capabilities and access control lists are shown to be extremes of this spectrum, each of which has its advantages and disadvantages. See also References 3 and 6.

*Following normal security conventions, active entities requesting access to other, protected entities in the system are referred to in this paper as subjects and the protected entities as objects.

2. THE SECURE SYSTEMS MODEL

2.1 Overview

In terms of the Open systems Interconnection (OSI) model, the level of view addressed here is application layer. The security entities described communicate using OSI services of sufficient security to satisfy their needs (Ref. 2). These needs take the form of guarantees, to some acceptable level, that communications between them and with their peers are confidential

and unmodified, and that each communication is with a known and identified peer entity.

Section 2.1.1 introduces the two level view of security necessary to distinguish properly between the network-wide security policy for a distributed system and the individual security policies support by the applications residing in the nodes of the network. Section 2.1.2 shows how the concept of a "subject" can change according to the nature of the accesses that are taking place.

2.1.1 Internal and External Application Views

There are two fundamentally different levels at which the security requirements of a distributed application network can be addressed:

- at the application access level, concerned with access to protected network objects like productive and supportive applications;
- at the application-specific level, concerned with access to objects supported by network applications, like files or documents.

These two levels of view have quite different requirements, reflected in different security policies tailored to the different kinds of protected objects involved and the different components of the network that are responsible for their support. Indeed entities that are considered protected objects at one level can become accessing subjects at another. This is illustrated in figure 1.

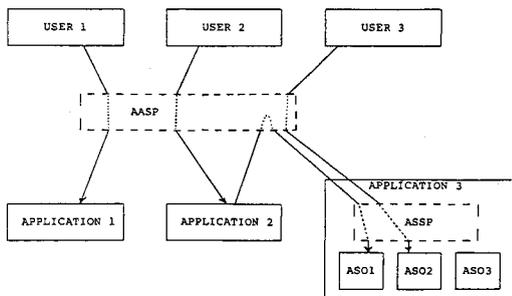


Figure 1 Network and Application Security Policies

Figure 1 shows a number of end-users wishing to access a number of network applications, policed by an Application Access Security Policy (AASP in the Figure). One of the applications is shown with its internal details revealed; it is supporting a number of protected Application-specific Objects (ASO1 to ASO3 in the Figure) being accessed both by end-users directly and by other applications in their own right, (viz: User 3 and Application 2) both being constrained by the same Application-Specific Security Policy (AASP in the Figure).

It is the support of the AASP that is the prime concern of TG9, since it is in this area that the distributed nature of a system is most apparent, and standard protocols are required to communicate security related information (e.g. the subject identity and access privileges discussed in Section 5) between applications running on end systems of different kinds and origins.

2.1.2 Indirect Access and Proxy

In some cases an application may be accessed by another application (for example Application 2 in Figure 1) rather than directly by a user. There are two possible extremes:

- the initiating application is acting on its own behalf;
- the application is acting on behalf of another subject (e.g. a human user).

The first situation may be used for example to restrict access to objects held on one application (say a File Service application) to requests coming via another (say a Database Service application). It is entirely appropriate for the Database Service to act with respect to the File Service as a subject with its own identity and access privileges. In this way end-user access to a protected object (in this case the database files) can be controlled in terms of the route and method used to access it. This kind of control is important to commercial organizations (Ref 9).

The second situation might occur for example when a user wishes to transfer a file directly from one application to another. The user requests one of the applications to initiate the transfer on his or her behalf. This application is acting as the user's proxy and must convince the other application that it is legitimately so doing before the transfer will succeed.

Hybrid cases can also exist in which the initiating application uses its own privileges in combination with those given to it by the calling user.

References 6 and 7 discuss proxy in more detail. Reference 7 examines the protocols involved in such situations.

3. SECURITY FACILITIES

At this stage no assumptions should be made about the degree of distribution of the facilities; this might vary from being a single central network application to being an aspect of every distributed supportive or productive application. Neither is it suggested that all of these facilities need be available on

every secure network. They should be viewed as a shopping list of items from which a choice can be made appropriate to the security policy and level of security quality required for the network. However, by identifying the full list, the framework causes omissions to be made evident and any resulting security weaknesses intentional rather than accidental.

The following security facilities have been identified:

3.1 User Sponsor

When a human user logs on to a distributed computer system using a (possibly remote) authentication facility (see 3.2) there is a requirement for local functionality that sponsors that user to the system, which controls the user's access to local applications and which monitors subsequent activity. The User Sponsor is the entity that provides these services.

There are two major security responsibilities that fall outside the ambit of particular productive applications. Firstly the User Sponsor is responsible for the monitoring of a user's access to local applications and which monitors subsequent activity. The User Sponsor is the entity that provides these services.

There are two major security responsibilities that fall outside the ambit of particular productive applications. Firstly the User Sponsor is responsible for the monitoring of a user's continued presence: no single application is in a position to time-out a user after a period of prolonged inactivity since the user may well have been fully active in other areas.

Secondly, the User Sponsor also

serves the user: it organizes the user's relationships with the various security facilities that come into play before, between and after his direct use of individual productive network applications. There is one instance of a User Sponsor for each active end-user.

User sponsors are further discussed in Reference 8.

3.2 Authentication

The Authentication Facility accepts and checks subject credentials, communicating its conclusions to other interested security facilities. The subject involved will either be an end-user via his or her user sponsor, a non-security application acting as a subject, or a non-security application coming on-line and making itself available as an accessible network object.

Notice that the authentication result is a proof of identity at an instant in time. Assurance of the continued validity of the mapping of this identity must be provided either by other means (e.g. time out by the User Sponsor), or by continued reauthentication in each subsequent data transfer.

3.3 Association Management

When a subject accesses an object a data exchange takes place. To provide the means by which this can happen, an association is established between them, and this must be established and maintained in a secure way. There are three aspects to this:

- Association management is responsible for ensuring that the underlying communications are as secure as is required by the communicating

entities, including assurance of their identities.

- The subject must have been authorized to communicate with the object. Association management must be sure either by the context within which the request was made, or by explicitly involving appropriate authorization facilities (see 3.6), that this is the case.
- Any security weaknesses inherent in the communications route chosen must be reflected in the access privileges of the subject. For example links on which there is no cryptographic protection should not be used for highly confidential data traffic, even though the accessing subject may be cleared to access such data.

3.4 Security State

The security state of a distributed system represents the current condition of the subjects and objects in it.

If a user is successfully authenticated then his or her condition, as recorded in the security state, changes; the same happens when a current file access is authorized or revoked or when a user logs off. The Security State Facility (SSF) is a passive facility that serves to hold a record of the current security state.

The SSF should not be confused with audit trail collection: SSF keeps the current state, not a record of state changes; however changes of security state will often be also recorded in an audit trail using an Audit Facility (see 3.8).

The SSF is an abstraction representing the state information of all of the elemental security facilities. It is therefore clearly likely to be highly distributed, with components in every node of the distributed system.

3.5 Security Attributes

The Security Attribute Facility provides appropriate subject-related access privilege data (such as a user's security clearances and group memberships) for already authenticated subjects, an object related access control data (such as its classification and access-control-list entries) for protected objects. A close relationship between the authentication facility and the security attribute facility is envisaged, particularly with respect to subject related privilege attributes. The data structures needed for both facilities are very similar, both being related to the structures defined by CCITT and ISO for Directories (Ref 10). TG9 has provided input to this work in connection with its proposals for security controls (Ref 11).

3.6 Authorization

The Authorization Facility uses access context, subject access privilege attributes and object access control attributes to authorize or deny requested accesses by subject to objects. The concept of authorization using privilege and control attributes is further discussed in Section 5.

3.7 Inter-Domain

If a security domain is defined as that part of a distributed system to which a single security policy applies under the responsibility of

a single security management entity, then special requirements arise when communication takes place between two security domains. In particular if a subject in one domain wishes to access a protected object in a second domain, additional rules are required which reflect the complex and varied trusted relationships that may exist between the different security domains. Domain A may or may not trust Domain B to authenticate its subjects, or may do so only to a limited degree. Some objects protected by Domain A may be so sensitive that no extra-domain access is permitted under any circumstances (Ref 4). Also Domain A's view of the meanings of particular security attributes may differ from that of Domain B, and finally, there may be a need to change cryptographic keys at the border between the domains. All of these matters require the support of an Inter-domain facility.

3.8 Security Audit

This facility provides security administrators with a record of the use of the security facilities of the system. It is the responsibility of other active security facilities to transmit audit information to the Security Audit Facility according to the audit policy for the system.

3.9 Recovery

This facility is available to a system administrator to take immediate corrective actions. These actions may come from a specific demand from the system administrator himself, or may be the result of events coming from the audit facility (alarms or security violations) or from other security facilities.

3.10 Cryptographic Support

Provides application layer cryptographic functions used both by other security support facilities and by productive applications to secure data in storage and transit in the following specific ways (see Ref 2):

- communications confidentiality
- communications integrity
- data origin authentication
- non-repudiation of origin
- non-repudiation of receipt

4. WALKTHROUGH

The following gives an example walkthrough of a user approaching the system and accessing an object controlled and supported by a productive application. The walkthrough is provided for illustrative rather than definitive purposes and not all the facilities are involved.

At the beginning of this walkthrough it is assumed that the system's security facilities are up and running with secure links established between them. The productive application of the example is already in service and authenticated as an accessible application object. The terminal's User Sponsor is also authenticated as legitimate (but no user is yet present). This implies that identities and addresses of these entities are now known to the Security State Facility.

A human user sees that there is a computer system in front of him or her. No other information is available (in this example). The user depresses a terminal key, puts in a magnetic badge or otherwise stimulates the system.

1. The User Sponsor is activated which connects the user to an Authentication Facility and mediates the authentication dialogue between them. The choice of Authentication Facility may or may not be made by the user. The user authenticates himself or herself and the Authentication Facility notifies this to the Security Audit Facility.

Similar notification actions occur also at other points in this walkthrough but are, from here on, omitted for reasons of clarity.

2. The Authentication Facility informs the Security State Facility of the successful Log-on, naming the User Sponsor involved. The Sponsor's identity is sufficient information to locate it.

3. The Authentication Facility asks the Attribute Facility for the authenticated user's access privilege attributes (possibly tempered by the authentication method used) and passes them to the Security State Facility to be remembered.

4. The user selects the required application.

5. The Association Management Facility is prompted by the User Sponsor to set up an association between the local and remote application entities on behalf of the user.

6. To do this, Association Management refers to the Security State and the Attribute Facility to obtain privilege and control attributes relating to the user, the service and the quality of association.

7. Association Management calls an Authorization Facility to check the user's right to access the

remote service. Association Management then sets up the association with the required quality of underlying security.

8. Having set up the association, appropriate changes are made by Association Management to the Security State. These changes may include further tempering of the user's privilege attributes based on the security quality of the association.

9. The User Sponsor informs the user that the connection has been made.

10. The user uses the newly established association to transmit a first request to the application to access an object supported by it.

11. An Authorization Facility in the productive application refers to Security State, specifying the association, so as to obtain both identity and access privilege attributes.

12. It then obtains the access control attributes associated with the object in question using an Attribute Facility within the application and uses them, in conjunction with the accessing user's privilege attributes to check the legitimacy of the access. The access is shown to be legitimate.

13. The user accesses the object!

Figure 2 shows the conversations, between the security facilities numbered using the numbering of the walkthrough description. The arrows point from initiator to responder in each case. All facilities may converse with an

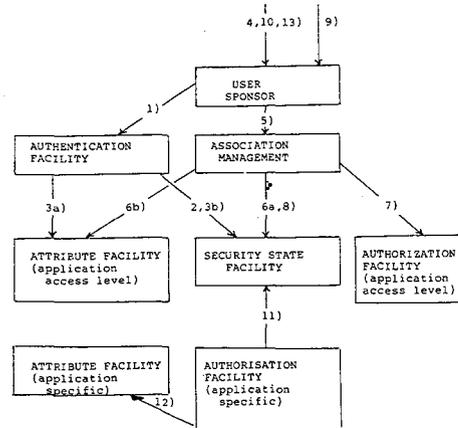


Figure 2 - Conversations between Security Facilities

Audit Facility or Recovery Facility.

5. THE AUTHORIZATION MODEL

5.1 Fundamentals

In the real world, authorization rulings are made in the context of characteristics possessed by the parties involved, the state of the world at the time, and the kind of access requested.

In the computer world we use similar concepts. The characteristics of the parties involved are represented by data which can be categorized as follows:

5.1.1 Authorization attributes associated with the subject (privilege attributes)

For example the subject's name(s), its role in the system and its trustworthiness. Indeed any attribute is a candidate for this category, provided that it is associated with the subject; in particular a name of an accessible object can be an attribute associated with a subject.

5.1.2 Authorization attributes associated with the object (control attributes)

For example the object's name(s), its role in the system and its degree of sensitivity or required integrity. Once again any attribute is a candidate for this category, provided that it is associated with the object. In particular a name of an accessing subject can be an attribute associated with an object.

5.1.3 The context within which the request is being made

For example the time of day, the communications route involved, or the accesses currently being made to other objects by this and other subjects.

Access contexts are not further considered in this paper, but are under study within the TG9 group.

5.1.4 The kind of access being requested

For example: read, modify, use, know-about.

The rules of the authorization policy are applied to values from these four categories and the result is essentially either "access permitted" or "access denied". The algorithm representing the rules is typically complex, involving complex combinations of multiple elements from each category. One of the tasks of the standardization process is to bring some structure to this complexity in a way that preserves as far as possible its general applicability. Notice that authorization attributes can be long lived or short lived. For example clearances and classifications tend to be static in nature, and therefore long lived. A capability on the other hand may be granted to a subject for the duration of a session, part of a session or for a single access.

Typically, authorization attributes are held as tuples, of which one part is the attribute's value and the other is one or more access types associated with that value. For example, if an object has associated with it an attribute containing the name of a particular subject, paired with an access-type value of "read", there is an obvious authorization rule that could be chosen to apply, under which presence of the attribute grants the subject read access to the object. Such an attribute would look remarkably like an access control list entry.

Not all attributes require this treatment however; for example a subject may have an attribute which defines its security clearance. Such an attribute will under many policies automatically be associated with read access since this is fundamental to the concept of security clearance. Such an association could therefore be made implicit.

5.2 Illustrative examples

5.2.1 If we imagine an object-name/access-type tuple as a privilege attribute (i.e. associated with a subject), with object-names also being associated with objects as control attributes, and couple these with an appropriate and obvious authorization rule we obtain what is essentially a capability.

5.2.2 If we imagine a "clearance" privilege attribute and a "classification" control attribute, and couple these with an appropriate authorization rule we have a label-based scheme which is appropriate for supporting a real world National Security Policy.

5.2.3 If we imagine subject name as a privilege attribute and a subject-name/access-type tuple as a control-attribute and couple this with an appropriate authorization rule we obtain what is essentially access via an Access Control List entry.

5.2.4 It is easy to devise more sophisticated variants of example

5.2.1 in which the object-name becomes an object type with more than one object possessing a given

'type' attribute, giving the capability a wider applicability. It is a small step further to consider this 'type' attribute as becoming a security label, and so arrive at example 5.2.2. A similar bridge could clearly be made between 5.2.2 and 5.2.3.

Thus clearances are revealed as generalizations of capabilities and classifications as generalizations of access control list entries.

Figure 3 illustrates this gradual merging of one concept into another. It includes also a bridge between 5.2.2 and 5.2.3.

EXAMPLE REF	SUBJECT PRIVILEGE ATTRIBUTE	OBJECT CONTROL ATTRIBUTE	NORMAL DESCRIPTION
5.2.1	OBJ NAME Access	OBJ NAME	Capability
5.2.4	OBJ TYPE Access	OBJ TYPE	Generalised capability
5.2.2	CLEARANCE ASSUMED READ	CLASS'N ASSUMED READ	Security Label
(5.2.4)	SUBJ GROUP	SUBJ GROUP ACCESS	Generalised ACL entry
5.2.3	SUBJ NAME	SUBJ NAME ACCESS	ACL entry

Figure 3 : The Authorisation Attributes Spectrum

5.3 Observations on the examples

5.3.1 The ease with which a capability mechanism can be transmuted into a clearance/classification mechanism and then into a conventional access control list mechanism argues for the usefulness and appropriateness of the underlying attribute framework.

5.3.2 When subject names are held at objects for use as control attributes (e.g. in ACL entries), day to day maintenance of the authorization policy is made difficult for systems with a volatile population of subjects. Conversely, when object names are held as privilege attributes associated with subjects (e.g. as Capabilities) maintenance is difficult for systems with volatile object populations.

Maintenance is therefore clearly a factor which should influence choice of expression of policy, and to define a standard for all systems based on one or other approach is consequently inappropriate.

Furthermore, a practical system is likely to require an authorization policy which uses multiple positions on the attribute spectrum. Figure 4 illustrates the point

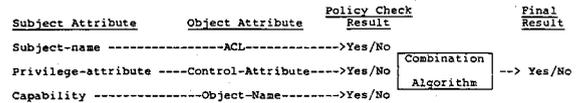


Figure 4 : Attribute Combination

Typically, high security systems might use an ACL approach for their discretionary authorization policy and a clearance/classification-attribute approach for the mandatory policy. A subject passing these tests can then (for performance reasons) be given a temporary capability which subsequently independently grants the requested access.

5.3.3 In a large distributed system, responsibility for control of an authorization policy might be devolved to a number of different centers. In particular, it will often be the case that control over the introduction of users to the network will be exercised by a different authority from those that administer the individual services on the network. The former could be considered to be the subject administrator of the network, and the latter the object administrators. On system with multiple cooperation authentication services there may be more than one subject administration authority.

It is useful to examine the authorization attributes that each authority controls. In general it seems obvious that subject administrators should be responsible for privilege attributes, with temporary attributes of either kind being allocated by object access control logic as implementation expedients. This fits in reasonably well with the real world perceptions of these attributes. It is entirely appropriate for a subject administrator to allocate user clearances, define which roles a user may assume and specify which department he or she belongs to. It is also appropriate for an object administrator to determine an object's ACL entries and security classification.

5.3.4 There are three levels at which standardization might be appropriate:

Level 1 - Define standard protocols for the passing of subject-related privilege attributes, confirming the definition to include only the means of protection and certification, the occasions when attributes are transmitted, and how they are obtained.

Level 2 - The definition of a set of standard attribute types within which the values used in real systems will fit (Ref 5).

Level 3 - A set of standard attribute values common to all conformant systems.

Level 1 standards would seem to be generally useful. There are parallels for Level 2 standards within the Directory proposals of OSI and CCITT (Ref 10), and a degree of commonality with these would be of value. Level 3 standards may be appropriate for a few widely used attribute types, particularly as used in protocol interactions with and between security support services.

5.4 Mandatory versus Discretionary authorization policies

A mandatory authorization policy is often defined as a policy based on security labels, with users possessing clearances like SECRET, and protected objects possessing similarly named classifications (e.g. Ref 13).

A discretionary authorization policy is in contrast thought of as a policy based on individual user identity, with users being granted or denied access on the basis of who they are rather than what clearance attributes are associated with them.

Under the authorization framework of this paper these differences are revealed as superficial; the labels of the mandatory policy and the subject-user identity attributes associated with capability or ACL approaches are merely variations on the same theme. Indeed, if under a mandatory policy users possessed unique non-hierarchic individual caveat clearances, the clearances become equivalent to user-id's and the corresponding classifications simplified ACL list entries.

Another distinction drawn between mandatory policies are centrally controlled, in contrast to the discretionary policy approach of control by ownership. In terms of the concepts of this paper, the difference lies in the allocation of access to the privilege and control attributes treated as projected objects. Looked at in this way, it becomes apparent that a variety of choices of devolution/centralization of control is possible, depending on the authorization policy associated with the attributes. This reflects the real world requirements exemplified by security manager, sub-manager, department manager, team leader, or individually based control policies.

A third difference drawn between mandatory and discretionary policies is that of rigor. In general, mandatory policies are expected to provide stronger protection for two main reasons:

- mandatory policies are usually implemented within an architecture which makes a clear distinction between trusted code and untrusted code. Policy control is ensured to be exercised only via trusted code, making evaluation easier and the level of assurance consequently higher.

- mandatory policies incorporate the concept of flow control (exemplified by the *-property of Ref 14, but more generally treated in Ref 15). This protects the system from malicious leakage of sensitive data to less sensitive environments by untrusted 'Trojan Horse' code.

In principle however there is no reason why a discretionary policy should not incorporate such features; in practice it is operational flexibility that determines the acceptability of constraining the software contexts within which control over the policy is exercised, and it is the granularity of the authorization policy that determines the ease or difficulty of policing information flows in a way which retains an acceptable degree of usability.

6. RELATIONSHIP TO THE DoD EVALUATION CRITERIA
(Ref 13)

It is not the task of the ECMA group to lay down criteria for assessing the strength of security in a distributed system, but the framework does provide a basis upon which such standards could be constructed. The major aim of the work however, is the definition of standards which will make independently designed network components able to work together in a secure manner.

The authorization model shows that the concept of mandatory versus discretionary control is an oversimplification; there is a complete spectrum of approaches of which the policies described in the DoD criteria are only two examples. Reference 9 lends further weight to this point.

In other respects, the framework and the detailed standards that grow from it will where relevant be developed to be compatible with the DoD criteria. The ECMA group regards the DoD criteria's requirement to separate trusted code from untrusted code as being a fundamental one, and the framework helps to define this separation by enabling trusted code functions to be identified and categorized.

7. CONCLUSIONS

The paper has described an application layer security framework which enables a distinction to be drawn between network-wide and local application security policies. A set of elemental security facilities has been defined and an example given of how these can work together.

Reference 12 includes a section by TG9 which sows some of these facilities combined into possible standard security applications. This work is continuing within TG9, and the group is looking towards the definition of standard communications protocols to and between standard security applications.

The paper has also described an authorization data structure which supports a variety of authorization mechanisms ranging from capabilities through label-based schemes to ACLs. It forms a basis for moving forward to develop standards relating to the kind of authorization data that is required to be passed between application entities in order to support their access control policies.

6. REFERENCES

1. "Application Integration: The Nature and Organization of Application Layer Standards" B. Wood, Proceedings of ONLINE '86 (Sept-Oct 1986).
2. "ISO 7498 Addendum-Security Architecture" ISO/DP 7498/2 (31 Oct 86).
3. "Issues in Discretionary Access Control" D. Downs et al., Proceedings of the 1985 IEEE Symposium on Security and Privacy.
4. "Non-Discretionary Controls for Inter-Organization Networks" D. Estrin, Proceedings of the 1985 IEEE Symposium on Security and Privacy.
5. "Attribute Transfer" T. A. Parker, ECMA/TC32-TG9/87/8 (Jan 1987).
6. "Authentication and Discretionary Access Controls in Computer Networks" P. Karger, Computers and Security 5. Pages 314-324 (1986).
7. "Third Party Transfer Security" T. A. Parker, ECMA/TC32-TG9/87/7.
8. "User Sponsors in A Secure System" A. C. Gale, ICL submission to ECMA TG9, (March 1987).
9. "A Comparison of Commercial and Military Security Policies" D D Clark and D R Wilson, Proceedings of the 1987 IEEE Symposium on Security and Privacy.
10. "Information Processing Systems - OSI - The Directory" ISO/DP9594/1-9 (23rd Oct 1986).
11. "Security of Directories" ECMA/TC32-TG9/87/12 (February).
12. "Framework for Distributed Office Applications" Final Draft ECMA/TC32-TG5/86 (December 1986).
13. "Department of Defense Trusted Computer System Evaluation Criteria" DoD 5200-28-STD (December 85).
14. "Secure Computer Systems: Unified Exposition and Multics Interpretation" D E Bell and L J LaPadula, ESD-TR-75-306, Mitre Corporation (March 1976).
15. "Cryptography and Data Security" D E Denning, Addison Wesley (1982).

APPLYING THE ORANGE BOOK TO AN MLS LAN

Dan Schnackenberg

Boeing Aerospace Company
Mail Stop 87-06
P.O. Box 3999
Seattle, WA 98124

This paper presents an overview of Boeing's multilevel secure (MLS) local area network (LAN) and a discussion of the issues that have arisen from applying the DOD Trusted Computer System Evaluation Criteria¹ (commonly called the "Orange Book") to this MLS LAN. Our MLS LAN has been designed and developed to meet the A1 criteria of the Orange Book, interpreted for a local area network, and is currently under developmental product evaluation with the National Computer Security Center (NCSC). A three node system is operating in our development laboratory to support integration, testing, and addition of new capabilities. This three node system utilizes prototype hardware, however, the initial product package is currently under development.

Our developmental product evaluation with NCSC began in late 1985 using the Orange Book as guidance in lieu of a network criteria. The current evaluation approach is to use the draft Trusted Network Interpretations (TNI)². In applying these interpretations, ensuring data integrity and preventing denial of service become issues.

MLS LAN OVERVIEW

Our MLS LAN is unique because of the number of services provided within the LAN. Figure 1 illustrates the objectives of the MLS LAN program. The MLS LAN provides both a back-end (host-to-host) network and a front-end (terminal-to-host) network, as well as interfaces to analog video and high bandwidth digital stream (e.g., digital sensors) devices. Wavelength division multiplexing is used on the fiber optic trunk to support simultaneous transmission of digital, analog video, and stream data.

The current capabilities include-

- Interterminal communication
- Terminal-to-host communication
- Reliable host-to-host communication
- Host-to-host datagram service
- Control of the physical circuit switching for analog video and high speed digital devices
- Comprehensive network management

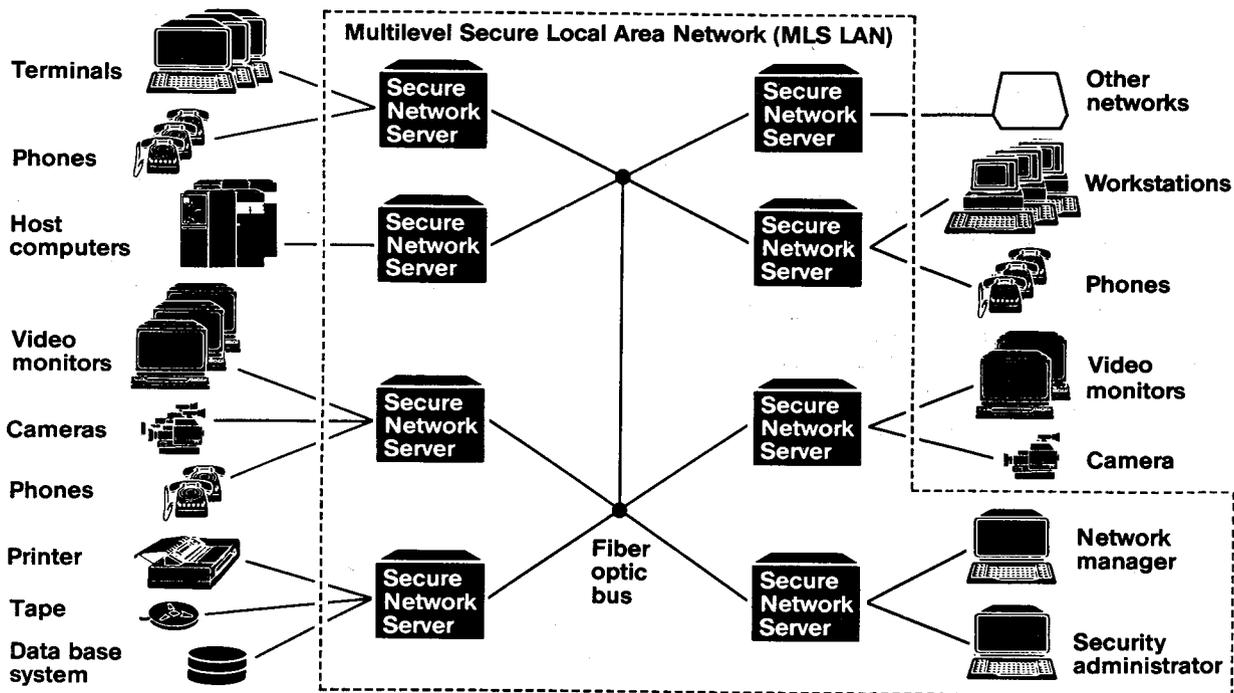


Figure 1: MLS LAN System Diagram

Future products are in varying stages of development. They include-

- File transfer support
- Simple Mail Transfer Protocol
- End-to-end encryption
- Gateway to the Defense Data Network
- MLS LAN bridge
- Alternate media access methods
- Extensive voice services
- Network mail
- File server
- Database server
- Printer server

The system is based on the DOD protocol suite, with full protocol support within the LAN for TELNET, Transmission Control Protocol (TCP), User Datagram Protocol, and Internet Protocol (IP). The IEEE standard 802.4 token bus protocol is used at the link layer.

The MLS LAN provides controlled access to the network medium by a variety of devices, including terminals, hosts, workstations, and video and stream devices (and eventually voice devices, printers, tapes, and disks) all within a multilevel environment. Our network management workstation provides centralized management of the network, while the Secure Network Servers (SNS) provide protocol processing and access control for the attached subscriber devices. Each device can be configured to operate within a range of sensitivity levels; and terminal, workstation, and host interfaces can be configured to support multiple concurrent sessions each operating at a different sensitivity level.

Within each SNS, a significant amount of software is required to support the range of user and security services. One of the A1 design objectives is to minimize the size and complexity of the network trusted computing base (NTCB). To meet this objective, the software within the SNSs is partitioned into both NTCB and non-NTCB components. Non-NTCB software provides protocol services, including TELNET, most of TCP, and most of the host-to-SNS protocol. Non-NTCB protocol functions provide many of the data integrity features addressed in the TNI. The NTCB functions ensure that non-NTCB processes supporting different user sessions cannot interfere with each other. Reference 3 provides a more detailed description of the software security architecture.

APPLYING THE TNI

In applying the criteria, our MLS LAN is evaluated as a single component. The MLS LAN comprises multiple devices: one or more SNS and a network management workstation. Each SNS contains one or more microprocessor. None of these

devices meets the A1 criteria by itself, however, the MLS LAN as a whole provides the features necessary to meet the A1 criteria. Most of the A1 criteria apply directly, without significant interpretation, to our MLS LAN. The following paragraphs discuss issues arising from the application of those criteria requiring interpretation.

Discretionary Access Control

In formally mapping the features of a packet-oriented network to a Bell-LaPadula-like model⁴, discretionary access control maps nicely to the requirement for correct addressing and delivery: only the sender of a packet and his protocol processes are permitted to write the packet, and only the addressee designated by the sender and his protocol processes are permitted to read the packet. The packet source and destination map to the discretionary access control matrix of the model for the packet object. In a connection-oriented system, the connection can be viewed as the object. For duplex communication, both participants in a TCP connection or TELNET session (and their protocol processes) are permitted read and write permission to the connection object. One-way communication is achieved by providing one of the participants with read-only access to the connection. This approach is used in our network to provide data transfer from lower security levels to higher security levels.

TCP's fully specified passive open provides an additional discretionary access control mechanism. The process requesting a passively opened socket may specify a remote socket, indicating that the requesting process wishes to only communicate with the remote process connected to the specified remote socket. Our network supports this feature and also permits the requesting process to specify the remote host without specifying the full remote socket. This is an extension to the TCP upper layer protocol interface supporting a capability similar to the specification of user groups for discretionary access control in operating systems. The NTCB services passive and active open requests, and provides the addressing and delivery functions at the link, network, and transport layers, which meets the interpreted requirements for discretionary access control.

For our physically circuit-switched services, we provide standard discretionary access control mechanisms. Users control circuit-switched devices and channels through the network's terminal interface. Users may request ownership of devices and channels, and may request that devices be connected to channels. When a channel is allocated to a user, the user is given the opportunity to specify a discretionary access control list for the channel. This list identifies the set of users permitted to connect receiving devices to the channel. This mechanism is similar to providing an access control list for files in an operating system, except that the only right that can be passed to other users is the right to receive.

Object Reuse

Within our LAN, we meet the standard requirements for object reuse: each storage object is set to a predetermined initial state before allocation to a non-NTCB process. The more difficult aspect of object reuse in our system is controlling reuse of distributed objects. One of the objects supported by our MLS LAN is the TCP connection. A connection is necessarily distributed between the two SNSs providing the connection object. From one connection to the next, the connection name is reused. For example, if a connection between sockets A and B is closed, and a new connection between the same sockets is opened, the new connection will have the same connection name as the old connection. The problem is ensuring that prior to reuse of the connection name, all remnants of the old connection are removed from the system ensuring that old connection data does not enter the new connection. This problem has led to the development of a session management protocol for initiating and terminating connections. This protocol is used between session managers at the two SNSs involved in the connection. Session managers are NTCB processes that control access to the connection objects. The major complicating factors in supporting this type of distributed object are (1) bit errors cause lost packets between the session managers; and (2) remote SNSs may have been shut down or reinitialized during the execution of the protocol, causing the session managers to lose synchronization. Our session management protocol addresses these problems.

Identification and Authentication

Identification and authentication of network users are required at the terminal interfaces to the LAN, however, users gaining access to the LAN through host computers are assumed authenticated by the hosts. Identification and authentication of hosts was determined to be not essential in the LAN environment. An SNS attached to a host will likely be collocated with the host, so that physical security for this interface can be assumed. Host identification can provide some protection against cabling errors, however, authenticating the host provides little assurance that the host is the expected host and has neither been penetrated, nor replaced.

A more reasonable requirement is for the network to detect disconnection of hosts and SNSs, and to forward this information to the network management workstation for display to operational personnel. Our network meets this requirement, providing network operational personnel assurance that they will be notified if the network configuration changes. This capability, plus physical security measures, provide reasonable assurance of the authenticity of a host's identity.

Trusted Path

The draft TNI requires a trusted path only from users to the NTCB. This is supported at our terminal user interface, ensuring that users are not spoofed by an application program masquerading as the NTCB. For host interfaces, there is a similar problem: the host needs mechanisms ensuring communication with the NTCB. This mechanism must support communication of labels, user identity, and addressing data between the host and SNS. Initialization and closing of the interface and user sessions are communicated using this "trusted path." The NTCB software that demultiplexes packets from the host implements this trusted path by scanning the protocol header to determine if the packet should be sent to a NTCB or a non-NTCB process. By sending packets with appropriate headers, the host is assured that the packet is received by the NTCB. These headers are also used by the SNS to mark packets from the SNS's NTCB. The SNS NTCB fills the headers preventing non-NTCB software in the SNS or a remote host from spoofing the host.

Audit

Audit requirements in a network differ significantly from those in hosts. The Orange Book requirement that "introduction of objects into a user's address space" be audited must be liberally interpreted to make sense in networks. For our connection-oriented services, the user's address space could be interpreted as including the address space of processes supporting the user connection in the SNSs. This would imply that all packet deliveries would have to be audited, creating significant audit overhead for the network. A more reasonable approach has been taken, requiring that connection events (creation and termination) must be audited, but not individual packet delivery.

System Integrity

For operating systems, off-line diagnostics are sufficient to meet the Orange Book system integrity requirements. System integrity requirements have been extended in the draft TNI to include mechanisms detecting loss of components. The 802.4 token bus protocol used in our MLS LAN provides the capability for SNSs to detect loss of a neighboring SNS. This information is forwarded to the network management workstation and displayed to network operational personnel. Each SNS is responsible for detecting loss of subscriber devices. These capabilities support both the system integrity requirements and the communications integrity requirements of the draft TNI.

Communications Integrity

Our MLS LAN provides several communications integrity

features to protect against transmission errors. Each SNS incorporates mechanisms providing assurance that (1) the remote session manager initiating the session is valid, (2) delivered packets do not contain errors, (3) packets for connections are delivered in order, and (4) packets are not lost.

Remote session manager authentication is implicit. Because the network can detect loss or addition of an SNS in the link layer protocol, each SNS that is currently on-line can be presumed valid. The security and network administrators are notified when SNSs enter and leave the token bus. An SNS must be on the token bus to transmit data intelligibly. This mechanism provides network administrators the capability to monitor the network configuration and identify the introduction of bogus SNSs. If network operational personnel adequately control the addition of SNSs to the token bus and physically validate the authenticity of SNSs when they are brought on-line, then the risk of a bogus SNS (and session manager) is minimal.

Protection of user data against modification is provided partially within the NTCB and partially in non-NTCB communication software. Our implementation of TCP places the communication integrity features outside the NTCB, including checksum, timeouts, retransmission, and packet sequencing. These features provide high assurance that the user data delivered is the same as the data transmitted, assuming that active wire-tapping is not present. Additional features are provided by NTCB hardware and software, including a 32-bit cyclic redundancy check at the link layer, the IP header checksum, and error-detecting memory in the SNSs, as well as features in the NTCB-to-NTCB protocol to ensure that NTCB data is delivered with high integrity.

Currently no protection is provided against active wire-tapping threats (e.g., playback and message modification) within our MLS LAN. The SNSs and transmission medium are assumed to be physically protected. We plan to address wire-tapping threats in future products through involvement in NSA's Commercial COMSEC Endorsement Program (CCEP).

Denial of Service

Denial of service protection within our MLS LAN includes mechanisms for (1) identification of the loss of components, (2) continued operation in the presence of component failures, (3) notification of network operational personnel when component failures are detected, (4) on-line reconfiguration of the network, and (5) network management controlled limitations of resource utilization to ensure that one user does not consume excessive amounts of critical resources denying service to other users.

Detection of the loss of components was discussed in the system integrity paragraph. Loss of a component affects only

the users of that component. The remainder of the network recovers automatically and continues operation. The exception is loss of the network management workstation. When SNSs cannot communicate with the network management workstation, they are designed to automatically shut down, which is required to meet the A1 criteria. The security administrator is provided the capability to override this feature and permit the network to continue in degraded mode when the network management node fails. This can be used in environments where continued operation is more critical than loss of audit data. Audit, terminal user login, circuit-switched services, and name service are lost when operating in degraded mode, however, terminal and host users can still communicate over existing sessions and can initiate new sessions provided the user does not require the name service feature. Design of a hot spare approach for network management, with automatic switch-over is in progress, but is not planned to be part of the initial product.

Several mechanisms are used to ensure that no user, or group of users, consumes an inappropriate share of the network's critical resources. At the lowest levels, within our MLS LAN's executive, a time-sliced scheduling discipline is enforced for non-NTCB processes. This ensures that each process has sufficient access to the CPU. NTCB processes are given as much time in the CPU as they need, while non-NTCB processes (i.e., those supporting user connections) are provided an equal share of the CPU. Each memory manager within an SNS uses memory quotas to prevent processes from using memory exhaustion to deny access to other users. Because multiple subscriber devices can be connected to a single SNS and the SNS has a maximum number of concurrent sessions that it can support, each subscriber device is allocated a maximum number of sessions that can be used by that device. Terminal users also have quotas limiting the number of concurrent sessions they are permitted. Finally, each SNS is provided a limit on how long it can transmit when it has the token, ensuring that no SNS transmits continually, denying access to the trunk to other SNSs. This "token hold time" can be used to allocate priorities to SNSs. An SNS is given access to a higher percentage of the trunk by being assigned a larger token hold time. The token bus protocol ensures that each SNS is provided an opportunity to transmit. Each of the quotas (memory, sessions, and token hold time) are set by the network administrator and can be modified on-line to reflect changes in priorities. Loss of service (denied access) because of resource exhaustion is an auditable event, which can be monitored by administrative personnel. These mechanisms do not prevent denial of service, but they do alert administrative personnel to denied access and provide administrative personnel the capability to prevent resource exhaustion by single users.

Network performance data is also accumulated and displayed to the network administrator. This provides the capability to determine when components of the system are

becoming overloaded causing degraded service to users. The network administrator can resolve the problem through reconfiguration of the network or modification of quotas to provide the affected users a larger share of the network resources.

STATUS

The developmental product evaluation of our MLS LAN is nearing completion. Most of the required documentation has been delivered to the NCSC, and with the release of the draft TNI, many of the uncertainties in the evaluation have been eliminated. The major issue remaining for the evaluation is to determine the impact of the latest TNI version.

The product development is also nearing completion. The major remaining tasks are (1) completion of the product from the existing advanced development models; and (2) completion of product testing. Production prototypes are expected to be completed during the first quarter of 1988. The product testing effort is underway.

REFERENCES

- [1] "DOD Trusted Computer System Evaluation Criteria," DOD 5200.28-STD. National Computer Security Center, Ft. George G. Meade, Maryland. December 1985. pp. 41-50.
- [2] "Draft Trusted Network Interpretations," National Computer Security Center, Ft. George G. Meade, Maryland. April 1987.
- [3] D. D. Schnackenberg, "Development of a Multilevel Secure Local Area Network," Proceedings of the 8th National Computer Security Conference, September 1985.
- [4] D. Bell and L. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation," MTR-2997, The MITRE Corporation, Bedford, MA, July 1975.

INFORMATION FLOW CONTROL IN A DISTRIBUTED OBJECT-ORIENTED SYSTEM WITH STATICALLY BOUND OBJECT VARIABLES

Masaaki Mizuno and Arthur E. Oldehoeft
Department of Computer Science
Iowa State University
Ames, Iowa 50011

A. INTRODUCTION

The modular approach to the design of computer systems has been the subject of considerable attention in the study of operating systems and programming languages. Fundamental characteristics to be enforced by modularization include encapsulation, data abstraction, synchronization of concurrent access, protection and reliability. Such characteristics are of special interest in an distributed network environment since access to common control data is not generally feasible.

For our work on protection, we have adopted a general object-oriented model. An object is an instance of an abstract data type in that it encapsulates object variables and operations on these variables. The object variables have an indefinite lifetime and may be shared by a community of users. Operations on these object variables are performed through the invocation of procedures which are exported by an object. This is the only means of inter-object communication. For purposes of this discussion, a underlying reliable multilevel security message passing system is assumed to exist.

Two types of securities are commonly considered: access control and information flow control. An access control policy specifies authorization for access to objects based on the identity of subjects. Information flow policy regulates the flow of information between classified objects. While a significant amount of research has been done on information flow control [2,8], the focus of attention in this paper is on the manner in which information flow policies can be enforced in an object-oriented distributed environment.

Since the various objects of program may be geographically distributed or constructed at different points in time, it may not be feasible for one object to access protection information of other objects. Instead, it is desirable to establish the "internal" information flow security of each procedure in an object independent of other procedures and other objects. Since this does not account for inter-object flow of information, the security of a program must be partially certified at run time. To be useful, this certification must be also efficient.

The method presented in this paper is a combined approach of compile-time and run-time information flow certification. The compile-time mechanism establishes the internal security of individual procedures and creates the necessary information structures to allow for efficient run-time certification of inter-object communication. The run-time mechanism completes the certification of the entire program at message passing time by verifying the information flow caused by procedure invocations.

B. A DEFINITION OF FLOW CONTROL

The underlying theory of information flow control is based on the lattice model $(SC, \leq, \oplus, \otimes)$ introduced by Denning [3], where

1. SC is a finite set of security classes;
2. \leq is a binary relation which induces a partial ordering on the security classes in SC ;
3. \oplus is an associative and commutative binary operator on SC , denoting the least upper bound, e.g. $A \oplus B$ is the least upper bound of security classes A and B ; and
4. \otimes is an associative and commutative binary operator on SC , denoting the greatest lower bound, e.g. $A \otimes B$ is the greatest lower bound of security classes A and B .
5. SC has the greatest lower bound LOW and the least upper bound $HIGH$ such that $LOW \leq A$ and $A \leq HIGH$ for all A in SC .

For notational convenience, if x is a variable, then the security class of x will be denoted by x .

An example of the use of such a security lattice occurs in military organizations where a security class is commonly designated as an ordered pair (classification, department). If a and b are classifications of information (e.g. UNCLASSIFIED, CONFIDENTIAL, SECRET, TOPSECRET) and x and y are compartments (representing need-to-know), then the partial order of the security classes is defined by

$$(a,x) \leq (b,y) \text{ if and only if } a \leq b \text{ and } x \subseteq y.$$

The policy governing secure information flow is determined by the security lattice. For simplicity, the examples in this paper assume a linear lattice of security classes consisting of UNCLASSIFIED (= LOW), CONFIDENTIAL, SECRET, TOPSECRET (= HIGH).

A program variable may be either statically or dynamically bound to a security class. A "statically bound variable" is assigned a fixed security class at the time of its definition. The security class of a "dynamically bound variable" changes with the class of its associated information. An information flow from variable A to variable B is denoted by $A \rightarrow B$. If B is a statically bound variable, then such a flow is secure if and only if the relation $A \leq B$ is implied from the lattice. Otherwise, a security violation occurs. Note that if B is a dynamically bound

variable, B becomes \underline{A} .

Flows can be classified as explicit or implicit. An explicit flow from variables a_1, \dots, a_n to variable x occurs when an execution directly assigns information derived from a_1, \dots, a_n to x . An implicit flow from variables a_1, \dots, a_n to variable x occurs when an execution of a statement which assigns some information to x is conditioned upon values derived from a_1, \dots, a_n . For example, the statement

$$\text{if } a > 0 \text{ then } x := y \text{ else } y := z$$

causes an explicit flow from y to x only when $a > 0$, and from z to y only when $a \leq 0$. The statement also causes an implicit flow from a to both x and y regardless of the value of a . Note that implicit flows occur even in absence of execution of statements. This will be illustrated in section D.

C. A REVIEW OF PREVIOUS INFORMATION FLOW MODELS

Information flow models can be characterized by

1. their ability to handle statically bound or dynamically bound variables, and
2. whether or not security is verified at compile-time or run-time.

Denning developed a compile-time certification procedure for programs with statically bound variables [4,5]. Certification rules are given for each statement type (e.g. assignment, if statement, while statement, etc.). One major difficulty of this approach, however, lies in handling of procedures. Since the class of all parameters must be statically declared, a different version of a procedure is required for each different security class of a parameter. This may not only be inconvenient but also severely impairs the flexibility of resource sharing. One possible solution for this problem is to disallow access to global variables. Under these restrictions, the output parameters are functions of the input parameters and the security of a procedure can be established by verifying

$$a_1 \oplus \dots \oplus a_m \leq b_1 \otimes \dots \otimes b_n$$

where a_1, \dots, a_m are actual **IN** parameters and b_1, \dots, b_n are actual **OUT** parameters of the call. The inability to effectively handle object variables is considered to be a major restriction.

In another model for dynamically bound variables, Denning extended Fenton's run-time approach [6,7] to account for implicit flows occurring even in the absence of the execution of statements [4]. The certification procedure relies on a hardware support mechanism which includes a tag field in each memory cell and a stack HS which contains the security class on which the currently executing statement is conditioned (to account for implicit flows). The class on the top of HS is denoted by \underline{HS} . The operation "push(e)" places " $\underline{HS} \oplus e$ " at the top of HS and the operation "pop" removes the class from top of HS. When an assignment statement

$$x = F(a_1, \dots, a_n)$$

is executed, the hardware automatically updates x to

$$a_1 \oplus \dots \oplus a_n \oplus \underline{HS}.$$

In order to account for implicit flows which occur in the absence

of the execution of statements, the compile time mechanism must also insert into the source program "update b " operations, which update HS to " $b \oplus \underline{HS}$ ". For example, the statement

$$\text{if } e \text{ then } S1 \text{ else } S2$$

is transformed to the code

```
push(e);
if e then S1 else S2;
for x in ((V1  $\cup$  V2) - (V1  $\cap$  V2)) do update x;
pop;
```

where $V1$ and $V2$ are sets of variables to which values are assigned in $S1$ and $S2$, respectively. The model relies on a compile-time analysis to insert push, pop and update operations and on run-time hardware to maintain the stack HS and tag fields in the memory cells. The drawback to this approach is that it requires special architecture and incurs significant run-time overhead.

Andrews and Reitman's developed a compile-time certification technique based on program verification [1]. Implicit flows are classified into two types: local flow and global flow. A local flow is an implicit flow within a statement. A global flow includes an implicit flow from the conditional variables of an iteration statement to all subsequent statements and also a flow caused by process synchronization. For example, the sequence of statements

```
x := 0;
while y > 0 do
x := 1;
```

causes a global flow from y to x since the last statement is conditionally executed, depending on the value of y .

In order to handle these two types of flows, special certification variables, local and global, are introduced. A value in local becomes

$$\underline{\text{local}} \oplus \underline{\text{exp}}$$

within a conditional statement, where exp denotes the class of the conditional expression. Upon completion of the conditional statement, the value in local reverts to its previous value. Global, on the other hand, represents an accumulation of classes of the conditions which would be in effect upon completion of the execution of body of a while statement or wait statement. For example, global becomes

$$\underline{\text{exp}} \oplus \underline{\text{local}} \oplus \underline{\text{global}}$$

immediately after a while statement. Note that global accumulates not only exp but also local in order to account for the case in which the while statement itself is nested within other conditional statements.

By using these certification variables, proof rules are presented for various types of statements, including synchronization statements (e.g. wait and signal). Variables may be either statically or dynamically bound. The verification of a procedure invocation requires previous verification of the body of the called procedure and previous establishment of the pre/postconditions.

Andrews and Reitman's model seems too restrictive for general distributed object-oriented systems in which dynamic linking is allowed. Also, the manner in which self/mutual recursive calls are verified is not clear. For our model, we need a certification mechanism which can verify the "internal" security of an object independent of other called objects, some of which may

not yet be certified or for which security information is not yet available.

D. THE INFORMATION FLOW MECHANISM

1. Overview

The method presented in this paper assumes dynamic run-time linking of inter-object procedure calls. Object variables are assumed to be statically bound while other variables may be either statically or dynamically bound. For most practical applications, this is a reasonable restriction. In addition, we seek an efficient method that performs as much of the certification work as possible at compile time and one which does not rely on special architectural features.

We assume the following syntax for a procedure invocation statement:

procedure PROC (**IN** x_1, \dots, x_l ; **OUT** y_1, \dots, y_m)

where the **IN** parameters are "call by value" and the **OUT** parameters are "call by result".

Our method incorporates and extends ideas from both Denning's and Andrews and Reitman's approaches. Its salient features are:

1. Object variables are statically bound. The classes of other program variables can either be dynamically or statically bound in order to eliminate the need for more than one version of an exported procedure.
2. Each procedure, exported by an object, can be compiled and its "internal" security established independent of other procedures.
3. For efficiency, run-time information flow security checks are performed only at message passing time.
4. Since object variables of an object have a lifetime which may exceed that of individual programs that call a procedure exported by the object, information flow control takes into account the security classes of these object variables.
5. **OUT** parameters of an exported procedure are not assumed to be a function of only **IN** parameters, that is each **OUT** parameter might actually be a function of some subset of the **IN** parameters and the object variables of this and other objects which are subsequently called.

In order to achieve these objectives, we use a combined compile-time and run-time method. At compile time, the internal security of individual procedures are established and the data structures used for efficient run-time certification of inter-object communication are generated. The certification of the entire program is completed at message passing time by verifying the information flow caused by procedure invocations.

Prior to explaining the method, we first identify all possible input and output values to/from a procedure in an object. We define the term "input variables" and "output variables" to stand for variables which carry input values to the procedure and output values from the procedure, respectively.

Possible input variables of a procedure PROC are:

- (1) formal **IN** parameters of PROC.
- (2) the object variables read by PROC, that is the values of the

object variables when the call is instantiated, and

- (3) actual **OUT** parameters returned from external procedures that are called by PROC.

Possible output variables of PROC are:

- (4) formal **OUT** parameters of PROC,
- (5) the object variables written by PROC, that is the values of the object variables when the call terminates, and
- (6) actual **IN** parameters to exported procedures in other objects that are called by PROC.

The purpose of the compile-time algorithm is to generate equations that express the potential run-time information flow in symbolic form. In order to do this, "symbolic class expressions" are generated for variables in terms of the classes of the input variables (1) (3). A symbolic class expression represents the class of information in terms of the classes of variables from which it is composed. For example, the class of information in the expression

$$A + B * C - D / E$$

is symbolically denoted by

$$A \oplus B \oplus C \oplus D \oplus E.$$

The classes of dynamically bound input variables cannot be determined until run time. During compilation, the classes of these input variables are established as "security variables". Security variables are symbolically denoted by

1. procedure-name.variable-name
(for formal **IN** parameters of the procedure being compiled), or
2. object-name.procedure-name.variable-name
(for actual **OUT** parameters of external procedures).

For example, if the procedure being compiled is F(**IN** a, b), then the classes of "a" and "b" are symbolically denoted by F.a and F.b, respectively. Also, if this procedure invokes a procedure G of an object O as O.G(**IN** x, **OUT** y, z), the classes of y and z are symbolically denoted by O.G.y and O.G.z, respectively.

Based on these symbolic class expressions, the compile-time algorithm generates two types of symbolic equations: a "symbolic class equation" and a "symbolic flow equation". A symbolic class equation is used to calculate the outgoing security classes of an output variable. One such equation is created for each actual parameter in (4) and (6), regardless of whether it is dynamically or statically bound. The equation has the form

$$\text{variable} \equiv \text{"symbolic class expression"}$$

which states that the information in "variable" has a security class given by the "symbolic class expression". A symbolic flow equation is used to check flow violations. One such equation is created for each statically bound variable (including object variables). The equation has the form

$$\text{variable} \equiv \text{security class} \leftarrow \text{"symbolic class expression"}$$

which states that the class of "variable" is statically bound to "security class" and the information whose class is given by "symbolic class expression" flows to "variable" during the execution. Both types of symbolic equations are stored in an "in-

formation flow template" in the object.

The distinct parts of an information flow template are:

EXPORT : This consists of symbolic class equations for the formal **OUT** parameters of the procedure.

IMPORT : This consists of symbolic class equations for the actual **IN** parameters of external procedures called by this procedure. Since there may be more than one externally invoked procedure, this part of the template consists of a list of all such procedure names, each of which is followed by equations for associated actual **IN** parameters. If the same procedure is invoked from N different places in the text, then N distinct procedures are assumed since the same procedure from different places could carry different sets of **IN** parameters, and consequently different security classes of the actual parameter values. (The formation of N distinct names could simply be carried out by a preprocessor prior to compilation.)

STATIC : This consists of symbolic flow equations for statically bound variables.

A security class is not associated with an object itself since flows are checked at the times that its exported procedures are invoked. Therefore, if an object is passed as a parameter or its identifier is stored in an object variable, the compile-time mechanism does not need to generate a symbolic class equation or a symbolic flow equation associated with the object. Flow control is carried out when exported procedures of the object are called.

An "information flow instance", based on the information flow template, is created at run time for each procedure invocation. The run-time certification mechanism completes the verification work when procedure invocation takes place. It is done by replacing the security variables in the information flow instance with actual security classes of the corresponding parameters carried by the message. If a procedure F calls another procedure G in another object, part of the verification of F may have to be deferred until G completes.

2. The Compile-Time Algorithm

a. Compile-Time Data Structures. The purpose of the compile-time algorithm is to generate information flow templates for exported procedures and the initialization procedures of objects. At the outset, the symbolic class expression for each program variable is initialized as follows:

1. For a statically bound variable (including a parameter), the class expression is defined to be its fixed security class.
2. For a dynamically bound local variable or formal **OUT** parameter, the class expression is defined to be **NULL**.
3. For a dynamically bound formal **IN** parameter, the class expression is symbolically represented by the corresponding security variable.

For efficiency purposes, reductions are performed on each symbolic class expression in order to yield a minimal form. Such a minimal form is either **NULL** or consist only of a fixed security class and zero or more security variables connected by "+" operators. Three reduction rules are involved:

1. Replace a symbolic class for a local or output variable in a symbolic class expression with the class expression representing the class of the variable.

2. Delete all duplicate security variables. For example,

$$a \oplus b \oplus a = a \oplus b.$$

3. Delete a fixed security class if a higher or equal class exists in the expression. For example,

$$\begin{aligned} \text{NULL} \oplus a &= a \\ \text{LOW} \oplus \text{HIGH} \oplus \text{LOW} &= \text{HIGH} \\ \text{SECRET} \oplus \text{TOPSECRET} &= \text{TOPSECRET}. \end{aligned}$$

The following example illustrates the reduction to minimal form for two successive assignment statements:

statement	symbolic class equation
$c := a + b + a$	$c = a \oplus b \oplus a$ $= a \oplus b$
$d := a + c$	$d = a \oplus c$ $= a \oplus a \oplus b$ $= a \oplus b$

The algorithm requires two special compile-time variables: a stack type variable **STACK** and a simple variable **GLOBAL**. **STACK** contains the security classes of the expressions on which the statement currently being analyzed is conditioned. Thus, **STACK** accounts for implicit flows (local flows, in Andrew and Reitman's model). As in Denning's notation, **STACK** denotes the class on the top of **STACK**, and the "**STACK.push(e)**" operation adds

$$\text{STACK} \oplus e$$

to the top of **STACK**. The "**STACK.pop**" operation removes the class on the top of **STACK**. **GLOBAL** holds a class of conditional expressions to reflect

1. implicit flows which will be in effect after completion of execution of "while" statements in the same manner as Andrew and Reitman's global, and
2. the implicit inter-object flow which will be flowing from a caller of a procedure **PROC** being certified (this implicit inter-object flow is denoted by security variable **PROC.implicit** and is explained in subsection D.2.c).

The class contained in **GLOBAL** is denoted by **GLOBAL** and is initialized to **PROC.implicit**.

The algorithm also uses compile-time array variables **SC** and **EXP** to form symbolic equations. The domain of **SC** is the set of all the variables which are used in a procedure being certified. The domain of **EXP** consists of all the statically bound variables used in the procedure. For a dynamically bound variable **x**, **SC[x]** is the symbolic class expression for **x**. The algorithm constructs the symbolic class equation

$$x = \text{SC}[x].$$

If **x** is a formal **OUT** parameter of the procedure being compiled, the equation is placed in the **EXPORT** category. If **x** is an actual

IN parameter of a procedure to be invoked in another object, the equation is placed in the IMPORT category of the information flow template. If a variable is statically bound, $SC[x]$ contains its fixed security class and $EXP[x]$ contains the corresponding symbolic class expression. The algorithm combines these two to construct the symbolic flow equation

$$x \equiv SC[x] \oplus EXP[x]$$

and places it in the STATIC category of the information flow template.

The compile-time algorithm is given in the Appendix. Subsequent subsections discuss some special semantic details.

b. Information Flow Semantics of Assignment. Assume an assignment statement of the general form

$$x := f(a_1, \dots, a_m).$$

If x is a dynamically bound variable, the algorithm generates

$$SC[x] = SC[a_1] \oplus \dots \oplus SC[a_m] \oplus \text{STACK} \oplus \text{GLOBAL}.$$

If x is a statically bound variable, the algorithm updates $EXP[x]$ as

$$EXP[x] = EXP[x] \oplus SC[a_1] \oplus \dots \oplus SC[a_m] \oplus \text{STACK} \oplus \text{GLOBAL}.$$

The following example explains why updating (instead of replacing) of class expressions is necessary for statically bound variables. Suppose A is a statically bound variable and initially $SC[A] = \text{CONFIDENTIAL}$ and $EXP[A] = \text{NULL}$. Assume statement S_i assigns the value of variable X to A and, later in the text, statement S_j assigns the value of Y to A . Using simple replacement, the symbolic class expressions generated for S_i and S_j would be

$$\begin{aligned} EXP[A] &= X \oplus \text{STACK} \oplus \text{GLOBAL}, \text{ and} \\ EXP[A] &= Y \oplus \text{STACK} \oplus \text{GLOBAL}, \text{ respectively.} \end{aligned}$$

If " $\text{STACK} \oplus \text{GLOBAL}$ " is PROC.implicit for both S_i and S_j and there are no other statements that assign values to A after S_j , the flow equation

$$A \equiv \text{CONFIDENTIAL} \oplus Y \oplus \text{LOW} \oplus \text{PROC.implicit}$$

would be constructed and placed in the STATIC category of the template. Assume, at run time, the classes of X , Y and the implicit inter-object flow are SECRET, CONFIDENTIAL, and LOW, respectively. The run-time certification algorithm would replace Y and PROC.implicit in the symbolic flow equation for A with CONFIDENTIAL and LOW, respectively and would certify the flow. Even though A holds CONFIDENTIAL information at the end of the execution, the program violates the flow policy by storing SECRET information (class of X) in variable A during the time period between the executions of S_i and S_j . Therefore, instead of replacement, the class expressions for statically bound variables must be accumulated using the \oplus operator in order to account for all possible information flows. The correct symbolic flow equation for A in the above example is

$$A \equiv \text{CONFIDENTIAL} \oplus X \oplus Y \oplus \text{LOW} \oplus \text{PROC.implicit}.$$

c. Information Flow Semantics of Conditional. For selection statements, the compile-time algorithm accounts for the possibility of executing either alternative. For example, in

the statement

$$\text{if } a > 0 \text{ then } x := b \text{ else } x := c.$$

the algorithm constructs the symbolic class expression " $x \equiv b \oplus c \oplus a$ ", accounting for the implicit flow from " a " and the explicit flows from both " b " and " c ". If a procedure call to another object " $R1$ " is conditioned upon some variable(s), then there is an "implicit inter-object" information flow. For example, in the statement

$$\text{if } a > 0 \text{ then } b := R1.f(x),$$

there is a flow from " a " to the local variables and object variables encapsulated by $R1$ (and objects called by procedures in $R1$, etc.). Since information flows across object boundaries are certified at run time, special treatment of these implicit flows is required.

To handle this implicit flow, the compile-time algorithm constructs a special symbolic class expression, denoted by implicit , which represents the accumulation of classes on which the procedure invocation is conditioned. implicit is actually the class of outgoing implicit inter-object flow and it has the form

$$SV_1 \oplus \dots \oplus SV_n \oplus \text{PROC.implicit}$$

where SV_i denotes the i th variable on which the invocation is locally conditioned and PROC.implicit denotes the class for the implicit inter-object flow incoming to procedure PROC from the previous calling object. implicit is stored with the corresponding procedure name in the IMPORT category of the template. (Thus, an entry in the template for each external procedure has a symbolic class equation for implicit as well as a symbolic class equation for each of its actual IN parameters.) At run time, a request for a procedure invocation causes the run-time algorithm to evaluate the corresponding implicit as the class of the outgoing implicit flow. This value, as well as the security class of each parameter, is attached to the message. Upon receipt of a message by a receiving object $R1$ for the procedure call request f , an information flow instance is created and the security class in the message for the implicit inter-object flow (now denotes incoming implicit flow) replaces security variable $f.\text{implicit}$ in symbolic class expressions in the information flow instance. Since $f.\text{implicit}$ of each procedure entry in the IMPORT category contains $f.\text{implicit}$, subsequent procedure invocation requests from this object to yet other objects carry an accumulated class of the implicit flow in implicit .

Implicit flows across object boundaries occur even when procedure invocations are skipped. Failure to check for such implicit flows can lead to undetected security violation. This can be clearly illustrated in a program with dynamically bound object variables. The example program in Figure 1 is adapted from [2]. Assume that actual IN parameter x to procedure h is bound to SECRET and takes a value either one or zero, dynamically bound formal OUT parameter y of h is initially bound to UNCLASSIFIED, and dynamically bound object variable Z in R and W in Q are initially bound to UNCLASSIFIED. First, assume x takes value the one. Since the invocation $R.f()$ is skipped, the value and the class of Z in R remain zero and UNCLASSIFIED. The invocation $R.g(\text{OUT } a)$ from h returns true and UNCLASSIFIED for a . Therefore, $Q.k()$ is invoked and W in Q becomes one and UNCLASSIFIED. Then, $Q.m(\text{OUT } y)$ returns one and UNCLASSIFIED for y . Now assume the value of x is zero. Since $R.f()$ is invoked, the value and the class of Z in

```

object P
  procedure h (IN x:integer.
              OUT y:integer):
    begin
      var a : boolean;
      y:=0;
      if x = 0 then R.f();
      R.g(OUT a);
      if a then Q.k();
      Q.m(OUT y);
    end;
end P;

```

```

object R
  state
    Z:integer;
  procedure f()
    begin
      Z := 1;
    end;
  procedure g(OUT y : boolean)
    begin
      if Z = 0 then y := true
      else y := false;
    end;
  initialize
    begin
      Z := 0;
    end;
end R.

```

```

object Q
  state
    W:integer;
  procedure k();
    begin
      W := 1;
    end;
  procedure m(OUT x : integer);
    begin
      x := W;
    end;
  initialize
    begin
      W := 0;
    end;
end Q.

```

Figure 1. Implicit Flows Across Object Boundaries

R become one and SECRET. Thus, the invocation R.g(OUT a) returns false and SECRET for a. Since Q.k() is skipped, W in Q remains zero and UNCLASSIFIED. Therefore, Q.m(OUT y) returns zero and UNCLASSIFIED for y. Note that after execution of h, y becomes equal to x. However, y erroneously remains UNCLASSIFIED.

In our model, the errors described in the previous paragraph can be prevented since all the object variables are statically bound to security classes. However, when procedures are invoked, the run-time algorithm must perform the following when an information flow violation is detected:

1. The violating procedure invocation is skipped,
2. The execution continues as if no flow violation were detected, and
3. The flow violation is not reported to the user. In this way, the user cannot discern between a skipped invocation and one that is in violation.

Suppose that in the above example, Z in object R and W in object Q are statically bound to class LOW. Then, the STATIC categories of the information flow templates for R.f() and Q.k() have the symbolic flow equations

$$\begin{aligned}
 \underline{Z} &\equiv \text{LOW} \leftarrow \text{LOW} \oplus \underline{f.implicit}, \text{ and} \\
 \underline{W} &\equiv \text{LOW} \leftarrow \text{LOW} \oplus \underline{k.implicit}, \text{ respectively.}
 \end{aligned}$$

Let x have class HIGH and value one, and let y have class LOW. Then, R.f() is not called, and the invalid flow of $x \Rightarrow Z$ is not detected. Thus, R.g(OUT a) returns true and LOW for a, and Q.k() is invoked with implicit = LOW. Since no flow violation is detected when Q.k() is invoked, W in object Q becomes one and consequently, Q.m(OUT y) returns one in y. If x has value one, then R.f() is invoked with implicit = HIGH. This invocation causes a flow violation. The invocation to R.f() is skipped, but the execution continues without reporting the violation to the user. As a result, Z in R remains zero and LOW, and consequently, y has value one. Therefore, the value of x cannot be deduced from the value of y. In general, Fenton proves that if

all variables are statically bound, security can be guaranteed by verifying only flows caused by execution of explicit assignments. However, the run-time algorithm must do the following when a flow violation is detected [7]:

1. The violating statement is skipped
2. Execution continues as if no flow violation were detected.
3. The flow violation is not reported to the user.

d. **Information Flow Semantics of Iteration.** Iterative constructs also require special consideration. Consider the example

```

a := x;
while a > 0 do
  begin
    R1.f1(IN a, OUT b);
    R2.f2(IN b, OUT a);
  end;

```

The first time the body of the loop is executed, the security class of actual IN parameter "a" for R1.f1 is \underline{x} . However, in subsequent iterations, the class of "a" is the security class of the OUT parameter value from R2.f2 determined in the previous iteration. Since the number of times the loop body will be executed is unknown at compile time, the compile-time mechanism must provide for verification of worst case information flow. This requires the simulation of iterations until the symbolic class expressions stabilize.

Without special provisions, the symbolic class equation for "a" would be

$$\underline{a} \equiv \underline{x} \oplus \underline{R2.f2.a}$$

and the run-time mechanism would simply replace R2.f2.a with the class of the OUT parameter value when the object receives a return message from the first invocation of R2.f2. But this would be incorrect since the security variable R2.f2.a would then disap-

pear from the symbolic class equation for "a" and the equation would not reflect the actual information flows from subsequent invocations of R2.f2. In order to correctly account for flows from procedure calls across all iterations, the run-time mechanism must add the classes of the return values to the symbolic class expression instead of replacing the security variables. In order to identify the procedures invoked within loops, the compile time mechanism attaches an accumulation flag (denoted by (*)) to the security variables for such procedures. Thus, the symbolic class equation for "a" in the above example is

$$a \equiv x \oplus \underline{R2.f2.a(*)}.$$

3. The Run-Time Algorithm

The run-time algorithm is invoked whenever an object sends or receives a message. Messages which are received by an object O are:

1. requests to invoke procedures which are exported by O and
2. return messages from the external procedures invoked by O.

Messages which are sent by an object O are:

1. requests to invoke external procedure in another object and
2. return messages from exported procedures which terminate in O.

When object O receives a request to invoke an exported procedure PROC, the run-time algorithm creates an information flow instance, which is a copy of the information flow template for PROC, and replaces security variables for the inter-object flow and all formal IN parameters with the corresponding actual security classes carried by the message. Note that if the security variable is marked with (*), the algorithm adds the actual security class to the class expression containing the security variable (rather than replacing it). The algorithm then checks for a flow violation in each flow equation in the STATIC category. If all the equations are certified to be secure, the request is accepted and a new process for PROC is initiated.

If PROC invokes an external procedure, say "O1.func1", execution of PROC is suspended. The run-time algorithm then looks up the entry corresponding to O1.func1 in the IMPORT category of the information flow instance and places the security classes of the corresponding actual IN parameters and implicit in the outgoing message.

In general, the symbolic class equation in the information flow instance corresponding to parameter x of O1.func1 has the form

$$x \equiv SV_1 \oplus \dots \oplus SV_m \oplus SC$$

where SV_i ($1 \leq i \leq m$) stands for the i th security variable and SC stands for a fixed security class. The security variables are ignored since they denote the security classes of variables which, at this point, are not yet flowing into x. Therefore, the algorithm only uses SC to determine the security class of actual parameter x. The assignments to x from the input variables corresponding to the security variables may occur later (in the case of loops)

or they have already been skipped and will never occur in this particular execution (in the case of if statements or already terminated loops).

When object O receives a return message from another (previously invoked) object, the algorithm replaces (in the information flow instance) every occurrence of the security variables for the actual OUT parameters with the corresponding security classes carried by the message. Then it checks for flow violations in each symbolic flow equation in the STATIC category. If no flow violation is detected, the message is accepted and the (suspended) calling process is resumed.

Upon normal termination of PROC, the algorithm looks up the EXPORT category of the information flow instance and attaches the security classes to the corresponding formal OUT parameters. After sending the return message to the calling object, the algorithm erases the information flow instance.

E. A PROGRAM EXAMPLE

A program, consisting of classes C1, C2 and C3, and the corresponding information flow templates are shown in Figure 2. Objects O1, O2 and O3 are instances of C1, C2 and C3, respectively. Class C1 defines a general program in the form of an initialization procedure. The object variable S1 is statically bound to the security class CONFIDENTIAL. The initialization procedure invokes f and g in objects O2 and O3, respectively. Since the initialization procedure is automatically invoked at object instantiation time, it does not have a formal caller and consequently the information flow template has no entry in the EXPORT category. Note that the symbolic class expressions in both the IMPORT and STATIC categories contain a term for accumulating the implicit inter-object flow from the object that instantiates O1. For this example, we will assume that the class of this implicit flow is LOW. Since O2.f is called within a loop, the security variable $O2.f.i$ corresponding to OUT parameter i, is marked with an (*).

Class C2 defines exported procedure f and object variable S2 which is bound to CONFIDENTIAL. f replaces the value in S2 with IN parameter x and returns the old value of S2 for OUT parameter y. Class C3 defines exported procedure g and object variable S3 which is bound to SECRET. g adds the square of IN parameter a to S3 and returns this value for OUT parameter b.

Assume that S1 and S2 have initial values 70 and 0, respectively. When O1 is instantiated, the information flow instance for the initialization procedure is created which is a copy of the information flow template for the procedure and the initialization procedure is automatically invoked. O2.f is called in the body of the while loop with actual IN parameter a (-140). The run-time algorithm determines the classes for the implicit inter-object flow and actual IN parameter a from the information flow instance (in this case, both are CONFIDENTIAL) and then attach these classes to the message. The underlying message passing system sends the message to object O2.

When O2 receives the message, the algorithm instantiates a new information flow instance based on the information flow template and replaces all the occurrences of f.implicit and f.x in the instance with CONFIDENTIAL. The information flow instance at this point is as follows:

```

CLASS C1
  state
    S1 : integer of security class CONFIDENTIAL;
  initialize
    var i , a : integer;
    begin
      i := S1;
      while i < 100 do
        begin
          a := i * 2;
          O2.f (IN a, OUT i)
        end;
        if i > 150 then O3.g (IN i, OUT S1);
      end initialize;
    end C1;

```

```

CLASS C2
  state
    S2 : integer of security class CONFIDENTIAL;
  procedure f (IN x : integer; OUT y : integer);
  begin
    y := S2;
    S2 := x
  end f;
  end C2;

```

```

CLASS C3
  state
    S3 : integer of security class SECRET;
  procedure g (IN a : integer; OUT b : integer);
  begin
    S3 := S3 + a * a;
    b := S3;
  end g;
  end C3;

```

The information flow template for C1.initialize

```

EXPORT
IMPORT
  O2.f (IN a, OUT i)
    implicit  $\equiv$  CONFIDENTIAL  $\oplus$  O2.f.i(*)
       $\oplus$  init.implicit
    a  $\equiv$  CONFIDENTIAL  $\oplus$  O2.f.i(*)  $\oplus$  init.implicit
  O3.g (IN i, OUT S1)
    implicit  $\equiv$  CONFIDENTIAL  $\oplus$  O2.f.i(*)
       $\oplus$  init.implicit
    i  $\equiv$  CONFIDENTIAL  $\oplus$  O2.f.i(*)  $\oplus$  init.implicit
STATIC
  S1  $\equiv$  CONFIDENTIAL  $\leftarrow$  CONFIDENTIAL
       $\oplus$  O2.f.i(*)  $\oplus$  O3.g.S1
       $\oplus$  init.implicit

```

The information flow template for C2.f

```

EXPORT
  f (IN x; OUT y)
    y  $\equiv$  CONFIDENTIAL  $\oplus$  f.implicit
IMPORT
STATIC
  S2  $\equiv$  CONFIDENTIAL  $\leftarrow$  CONFIDENTIAL
       $\oplus$  f.x  $\oplus$  f.implicit

```

The information flow template for C3.g

```

EXPORT
  g (IN a; OUT b)
    b  $\equiv$  SECRET  $\oplus$  g.implicit
IMPORT
STATIC
  S3  $\equiv$  SECRET  $\leftarrow$  SECRET  $\oplus$  g.a  $\oplus$  g.implicit

```

Figure 2. An Example Program

```

EXPORT
  f(IN x; OUT y)
    y  $\equiv$  CONFIDENTIAL
IMPORT
STATIC
  S2  $\equiv$  CONFIDENTIAL  $\leftarrow$  CONFIDENTIAL

```

Since the flow equation for S2 in the STATIC category guarantees this to be a secure invocation, the execution of f is initiated. The value of S2 becomes 140 and OUT parameter y becomes 0. Also, the class of y (= CONFIDENTIAL) is determined from the information flow instance. Upon the termination of the execution, the algorithm erases the information flow instance and the message passing system sends the return message to O1.

When O1 receives the return message, the algorithm adds CONFIDENTIAL (the class of the return value i) to all the

symbolic class expressions which contain O2.f.i(*) in the information flow instance forming

```

EXPORT
IMPORT
  O2.f(IN a, OUT i)
    implicit  $\equiv$  CONFIDENTIAL  $\oplus$  O2.f.i(*)
    a  $\equiv$  CONFIDENTIAL  $\oplus$  O2.f.i(*)
  O3.g(IN i, OUT S1)
    implicit  $\equiv$  CONFIDENTIAL  $\oplus$  O2.f.i(*)
    i  $\equiv$  CONFIDENTIAL  $\oplus$  O2.f.i(*)
STATIC
  S1  $\equiv$  CONFIDENTIAL  $\leftarrow$  CONFIDENTIAL
       $\oplus$  O2.f.i(*)  $\oplus$  O3.g.S1

```

and the information flow is certified. (Note that if the class of

OUT parameter y of the return message had been SECRET, the flow equation for S1 would have become

$$S1 \equiv \text{CONFIDENTIAL} \leftarrow \text{SECRET} \oplus \underline{O2.f.i(*)} \oplus \underline{O3.g.S1}$$

and the run-time algorithm would have detected the flow violation.)

After the resumption of the execution in O1, i becomes 0 and the body of the while loop is again executed. This time, O2.f is called with actual IN parameter a (= 0). The message carries the value of a, the class of a (= CONFIDENTIAL) and the class for the implicit inter-object flow (= CONFIDENTIAL).

The algorithm certifies the information flow to procedure f of O2 and upon completion of execution, f returns y (= 140) and its security class (= CONFIDENTIAL) to the initialization procedure of O1. Since the conditional expression of the while statement is false, the loop terminates. Finally, since the conditional expression of the if statement is false, the execution terminates normally.

As a second example, suppose the initial value of S1 is 80. Again, the while statement terminates normally after the second iteration. However, i has the value 160 in the if statement and O3.g is invoked. The algorithm places the class of i (= CONFIDENTIAL) and the class of the implicit inter-object flow (= CONFIDENTIAL) in the message. Upon a receipt of the message at O3, the algorithm instantiates the information flow template for g and replaces all the occurrences of g.a and g.implicit with CONFIDENTIAL forming

EXPORT

g(IN a: OUT b)
b ≡ SECRET

IMPORT

STATIC

S3 ≡ SECRET ← SECRET.

Since the flow to S3 is certified, the execution is carried out. After g terminates normally, the return message is constructed which contains the value of OUT parameter b and its security class SECRET and the message is sent to O1.

In O1, the algorithm replaces O3.g.S1 with SECRET and the information flow instance becomes

EXPORT

IMPORT

O2.f(IN a, OUT i)
implicit ≡ CONFIDENTIAL ⊕ O2.f.i(*)
a ≡ CONFIDENTIAL ⊕ O2.f.i(i)

O3.g(IN i, OUT S1)
implicit ≡ CONFIDENTIAL ⊕ O2.f.i(*)
i ≡ CONFIDENTIAL ⊕ O2.f.i(*)

STATIC

S1 = CONFIDENTIAL + SECRET ⊕
O2.f.i(*) (** flow violation **)

The algorithm detects a flow violation. However, as mentioned in section D, the system cannot report the error to the user and, since the invocation causing the error is the last statement of the initialization procedure of O1, the execution must be terminated as if nothing has happened. Otherwise, one bit of information (the truth or falsity of the conditional) is sent to the user terminal (output file). This could be an undetected flow violation depending on the class of i and the clearance of the user.

F. CONCLUSION

This paper has presented an information flow certification mechanism for an distributed object-oriented system. The mechanism is a combination of compile-time analysis and run-time certification with the following salient features:

1. Information flow security checks are done only at message passing time.
2. Object variables encapsulated by an object are statically bound to security classes. Other program variables can be either dynamically or statically bound to security classes.
3. Each exported procedure in an object can be compiled and its "internal" security established totally independent of other exported procedures.

Information flow semantics were presented for selected programming constructs. Work in progress consists of extending the algorithm to allow for dynamically bound object variables. We are also investigating different ways to cope with the problem of illegal information flow from variables in a conditional expression to the user, caused by system generated error messages.

References

- [1] G. R. Andrews and R. P. Reitman. An axiomatic approach to information flow in programs. *ACM Transactions on Programming Languages and Systems*, 2(1):56-76, 1980.
- [2] D. E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.
- [3] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236-243, 1976.
- [4] D. E. Denning. *Secure Information Flow in Computer Systems*. PhD thesis, Purdue University, 1975.
- [5] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504-512, 1977.
- [6] J. S. Fenton. *Information Protection Systems*. PhD thesis, University of Cambridge, 1973.
- [7] J. S. Fenton. Memoryless subsystems. *Computer Journal*, 17(2), 1974.
- [8] C. E. Landwehr. Formal models for computer security. *Computing Surveys*, 13(3):247-278, 1981.

APPENDIX: The Compile-Time Algorithm

This section describes the compile-time algorithm

form information flow template

which generates an information flow template for exported procedures of an object. The following programming constructs are assumed:

1. declaration statement (the declaration of exported procedures and local variables),
2. assignment statement,
3. compound statement,
4. if statement,
5. while statement
6. procedure invocation statement, and
7. end statement of the procedure declaration.

This algorithm is applied to each statement of a procedure being compiled. STACK, GLOBAL, STATIC and TEMPLATE are global compile-time variables. The following initialization is done prior to its application to an exported procedure in an object:

1. The entries in SC and EXP for each object variable y are created and initialized as

$SC[y] =$ security class of the object variable
 $EXP[y] = \text{NULL}$
 $STATIC[y] = \text{true}$.

2. The preprocessor renames invocations of the same external procedure from different places in the text in order to make all calls distinct. Then entries corresponding to all the externally invoked procedures are created in the IMPORT category of TEMPLATE. For example, if the procedure being compiled is PROC and an external procedure $O.g(\text{IN } x_1, \dots, x_m, \text{OUT } x_{m+1}, \dots, x_n)$ is invoked in PROC, then these entries are

$O.g(\text{IN } x_1, \dots, x_m, \text{OUT } x_{m+1}, \dots, x_n)$
 $\text{implicit} \equiv \text{NULL}$
 $x_1 \equiv \text{NULL} \quad \vdots$
 $x_m \equiv \text{NULL}$.

3. GLOBAL and STACK are initialized to PROC.implicit and LOW, respectively.

The algorithm is stated below.

procedure form information flow template

(S : statement;

var SC, EXP : array of symbolic class expression;

var V : set of variables; LOOP : boolean);

var

V1, V2 : set of variables;

SC1, SC2, EXP1 : array of symbolic class expression;

CHANGED : boolean;

CE, temp : security variable;

begin

case S of

S = "procedure PROC

(IN x_1 : var.type of security class C_{x_1} ;

:

x_k : var.type of security class C_{x_k} ;

OUT y_1 : var.type of security class C_{y_1} ;

:

y_m : var.type of security class C_{y_m})"

begin

for $i := 1$ to k do

if x_i has a " C_{x_i} " declaration part

then

begin

$SC[x_i] := C_{x_i}$;

$EXP[x_i] := \text{PROC}.x_i$;

$STATIC[x_i] := \text{true}$

end

else

begin

$SC[x_i] := \text{PROC}.x_i$;

$STATIC[x_i] := \text{false}$

end;

for $i := 1$ to m do

if y_i has a " C_{y_i} " declaration part

then

begin

$SC[y_i] := C_{y_i}$;

$EXP[y_i] := \text{NULL}$;

$STATIC[y_i] := \text{true}$

end

else

begin

$SC[y_i] := \text{NULL}$;

$STATIC[y_i] := \text{false}$

end;

end;

```

S = "var  $a_1$  : var type of security class  $C_{a_1}$  :
    :
     $a_r$  : var type of security class  $C_{a_r}$ ;"
begin
  for  $i := 1$  to  $r$  do
    if  $a_i$  has a " $C_{a_i}$ " declaration part
      then
        begin
           $SC[a_i] := C_{a_i}$ ;
           $EXP[a_i] := NULL$ ;
           $STATIC[a_i] := true$ 
        end
      else
        begin
           $SC[a_i] := NULL$ ;
           $STATIC[a_i] := false$ ;
        end;
      end;
    end;
end;

```

```

S = "b := f( $a_1, \dots, a_m$ );"
/* Assignment Statement */
begin
  if  $STATIC[b]$ 
    then
      begin
         $V := 0$ ;
         $EXP[b] := EXP[b] \oplus SC[a_1] \oplus \dots \oplus SC[a_m]$ 
           $\oplus STACK \oplus GLOBAL$ 
      end
    else
      begin
         $V := \{b\}$ ;
         $SC[b] := SC[a_1] \oplus \dots \oplus SC[a_m]$ 
           $\oplus STACK \oplus GLOBAL$ 
      end;
    end;
end;

```

```

S = "begin  $S_1; \dots; S_m$  end"
begin
   $V := 0$ ;
  for  $i := 1$  to  $m$  do
    begin
      form information_flow_template
        ( $S_i, SC, EXP, V1, LOOP$ );
       $V := V \cup V1$ ;
    end;
  end;
end;

```

```

S = "if E then  $S_1$  {else  $S_2$ }"
begin
   $STACK.push(E)$ ;
   $SC1 := SC$ ;
  form information_flow_template
    ( $S_1, SC1, EXP, V1, LOOP$ );
  if  $S2 \langle \rangle 0$ 
    then
      begin
         $SC2 := SC$ ;
        form information_flow_template
          ( $S_2, SC2, EXP, V2, LOOP$ );
      end
    else
       $V2 := 0$ ;
  if x is in ( $V1 \cap V2$ )
    then  $SC[x] := SC1[x] \oplus SC2[x]$ 
    else
      if x is in ( $V1 - (V1 \cap V2)$ )
        then  $SC[x] := SC1[x] \oplus SC[x]$ 
        else  $SC[x] := SC2[x] \oplus SC[x]$ 
      end;
   $STACK.pop$ 
end;

```

```

S = "while E do  $S_i$ ;"
begin
   $CE := NULL$ ;
  repeat
     $CHANGED := false$ ;
    if  $CE \langle \rangle (CE \oplus E)$ 
      then
        begin
           $CE := CE \oplus E$ ;
           $CHANGED := true$ 
        end;
     $STACK.push(CE)$ ;
     $SC1 := SC$ ;
     $EXP1 := EXP$ ;
    form information_flow_template
      ( $S, SC1, EXP, V, true$ );
    if  $SC \langle \rangle (SC \oplus SC1)$ 
      then
        begin
           $SC := SC \oplus SC1$ ;
           $CHANGED := true$ 
        end;
     $STACK.pop$ ;
  until  $EXP = EXP1$  and not  $CHANGED$ ;
   $GLOBAL := GLOBAL \oplus LOCAL \oplus CE$ ;
end;

```

```

S = "O.g(IN x1, ..., xk, OUT y1, ..., ym)"
begin
  On O.g entry in the IMPORT category
  begin
    add " $\oplus$  STACK  $\oplus$  GLOBAL" to implicit;
    for all x in IN parameters of O.g do
      if not STATIC|x|
      then
        add " $\oplus$  SC|x|" to the symbolic
          class expression for x;
      else
        if the entry for x is " $x \equiv \text{NULL}$ "
        then replace that with " $x := \text{SC|x|}$ ";
      end;
    end;
    V :=  $\emptyset$ ;
    for all y in OUT parameters of O.g do
      begin
        if not STATIC|y| then V := V  $\cup$  {y};
        if LOOP
          then temp := O.g.y(*)
          else temp := O.g.y;
        if STATIC|y|
          then EXP|y| := EXP|y|  $\oplus$  temp  $\oplus$  STACK
             $\oplus$  GLOBAL;
          else SC|y| := temp  $\oplus$  STACK  $\oplus$  GLOBAL;
        end;
      end;
    end;
  S = "end PROC"
  begin
    for all x such that "STATIC|x| = true" do
      if EXP|x|  $\neq$  NULL then
        place " $x := \text{SC|x|} := \text{EXP|x|}$ "
          in the STATIC category
          of TEMPLATE;
      for all x in OUT parameters of PROC do
        place " $x := \text{SC|x|}$ " in the EXPORT category
          of TEMPLATE;
      end;
    end;
  end case;

```

end form information flow template.

THE ARCHITECTURE OF A DISTRIBUTED TRUSTED COMPUTING BASE

Jon Fellows, Judy Hemenway, Nancy Kelem, and Sandra Romero*

Unisys **
2525 Colorado Blvd.
Santa Monica, CA 90405

ABSTRACT

This paper explores the differences between monolithic and distributed Trusted Computing Bases, using as an example an actual system now in the final stages of development. For each of the differences discussed, the approach taken in the system is briefly described and motivated. The paper includes a description of the security policy of the system and its correspondence to the Bell and LaPadula model.

1. BACKGROUND

The need for trusted computing systems which process data at multiple security levels is widespread in defense related programs. Such systems have been an active research area for over ten years, with the result that worked examples of tightly coupled*** multi-level secure systems have been demonstrated [Fra83, Whit74]. A set of criteria for the architecture of multi-level systems has been established by the DoD Computer Security Center [DoD85]. These criteria, known popularly as the "Orange Book", also address the assurance that must be provided that the architectural criteria have been met. At the highest Orange Book category, A1, formal specification and verification are required at the design and policy levels.

The requirement for handling data at multiple levels goes beyond the usual operating system concern of local users sharing local resources; it is also being imposed on a current generation of embedded distributed systems. Even though no worked examples of secure distributed systems exist, the same criteria are being applied in the belief that such systems are a natural extension of the previous work on tightly coupled operating systems. This paper reports on the security architecture of one of the first secure distributed systems to be attempted: the issues raised, the approaches taken, and the lessons learned.

1.1 Terminology and Basic Concepts

A Multi-Level-Secure (MLS) computer system protects information on the basis of security labels which are attached to the

* This paper presents the opinion of its authors, which is not necessarily that of Unisys or of the Department of Defense.

** Formerly System Development Corporation

*** Tightly coupled is used here in the sense that the system can be modelled as a single state machine.

components of the system. System components include both data objects and active components of the system; but a given component may play both roles at different times in its lifetime. For the purposes of this discussion labels are assumed to be associated with a component at the time it is created and to retain their initial values for the life of the component. As is usual, components themselves have components, leading to a hierarchical structure that spans 'system' to individual variables and program statements. The granularity with which a system protects information is determined by the level(s) of components that carry labels. All current MLS systems cease explicit labelling at some point in the component hierarchy, with the result that lower level components inherit implied labels.

Each active component of the system, e.g. a process, a service, a subsystem, has a domain of execution which defines the set of data objects to which it may potentially be granted access. "Access" has system-specific meaning, but current systems focus on "read access" and/or "write access". Read access describes any system defined interaction between components in which information flows from a data component to an active component, while write access describes the converse. The domain of an active component can be further broken down into read and write domains, which will normally be expected to overlap. Active components interact either by sharing data objects across domains (e.g. a shared file), or by exchange of data objects between domains (e.g. a message system or input/output).

A system is MLS if all interaction between components preserves, with some level of assurance, the confinement of data objects to the read and write domains of active components with compatible labels. A confinement failure is known as a compromise. The compatibility of active component labels and data component labels is determined by the following domain confinement rule:

A component (either active or data) may potentially receive information from another component marked at any level "dominated by" its own label, where "dominates" is a system specific partial ordering of labels. This implies that the label of an active component must dominate the labels of each data component in its read domain; and also that the label of an active component must be dominated by the label of each data component in its write domain.

The domain confinement rule restates the basic properties of the well-known Bell and LaPadula model [BLP76]: the simple security property and the *-property.

It is important to note that the domain confinement rule constrains the domains of active components from containing data components with the potential for compromise, regardless of the actual compromise that the component might cause in a less constrained domain. The actual behaviour of the component could be compromise free in the less constrained domain, depending on its internal logic and its actual (as opposed to potential) pattern of references to other components. It would even be possible for a component to be compromise-free with respect to its domain in one given state, while causing compromise in domains with other states. An active component is called compromise correct if it is compromise free in all possible domains in which it can function as part of the overall system. Compromise correct components can be exempt from the domain confinement rule without changing the MLS-ness of the system.

A Trusted Computing Base (TCB) is the set of system components which, in order for the system to be MLS, must function correctly in the roles they play in the system architecture. In principle the TCB can encompass all of a system's components, but it is very costly to provide assurance of confinement using this approach; since each component, and each interaction between components, must be examined. In other words, each component must be compromise-correct. In practice, this approach is limited to small dedicated systems with a static set of components. For larger systems, particularly those which are open to the introduction of new components by untrusted users, a better approach isolates a small subset of components into a Reference Monitor [And72] which enforces the domain confinement rule. The TCB then becomes the reference monitor components and a small set of trusted components which are compromise correct.

In order to prevent untrusted components from interfering in the correct execution of reference monitor code, it is customary for a reference monitor to have a privileged domain of execution which includes not only the domains of all subjects but additionally contains objects not in the domain of any subject in the system. These reference monitor private objects normally contain significant portions of the system security state such as labels, clearances and passwords. In some implementations, reference monitor private objects are not themselves labelled.

2. WHY DISTRIBUTED TCBS ARE DIFFERENT

This section discusses the difference between a traditional monolithic TCB, where all TCB components share a domain and communicate by shared variables and procedure calls; and a distributed TCB, where TCB components are distributed over a network and communicate by exchanging messages. All of the classical TCB security issues must be addressed by a distributed TCB; but some issues, such as formal verification of correctness, are made more difficult by distribution of TCB components. Other issues,

such as trusted paths between TCB components, are new to distributed TCBS (or at least have been implicit in previous models).

The increase in complexity that results from distributing a TCB forces increased reliance on architectural arguments for security assurance, due to the weaker assurance possible from formal arguments. Perhaps this is only more evident for the distributed TCB case than has been the case for single domain TCBS. We have always relied on architectural arguments for hardware assurance of domain separation, and for locally reliable storage and transmission of data. Analogous functions in a distributed TCB may be implemented in software, but they are no less complex nor easier to verify.

The following subsections explore the five primary differences that we have been able to identify as requiring extra attention when building a distributed TCB. Briefly, they are fragmentation of the TCB domain, trusted paths, trusted protocols, hierarchical TCBS, and fault tolerance.

2.1 Fragmented TCB Domain

In a monolithic TCB the concept of a secure state can be expressed by an integrity constraint on the values held by security relevant data objects within the TCB's domain, for example that current accesses of subjects to objects are consistent with a security policy based on the security labels associated with those subjects and objects. All components of the security state are immediately available and stable in their values. It is possible for the monolithic TCB to guarantee that the security state changes one well defined step at a time and that after each change the security state meets its integrity constraints.

In a distributed TCB the security state of the system, rather than being collected into a single protected domain, is distributed across various devices of the system. Maintaining integrity constraints on a distributed security state is complicated in the following ways:

- a. It is difficult to check the system security integrity constraints at a single device, since remote components of the system security state are not obtainable without delay and are not guaranteed stable. It may still be possible to assert meaningful integrity constraints, but this situation causes an overall weakening of the constraints that can be asserted.
- b. Instead of a totally ordered sequence of state transitions, a distributed TCB's state history is only partially ordered due to the possibility of concurrent transitions at different devices. Again, it is still possible to state meaningful integrity constraints, but a careful analysis of potential interference between concurrent transitions is needed.

- c. Parts of the security state may be replicated at several devices in order to increase system availability. The system security constraints must then assert consistency of values for all replicated components of the security state regardless of where they are stored.

Although not normally viewed as part of a system's security state, the types and operations defined within the TCB can be viewed as data values of a distributed TCB which are replicated everywhere they are used. The best example of this is the type that defines the values held by security labels; if this definition varies from site to site within the TCB then a meaningful definition of policy enforcement is not possible. Likewise the operation that computes the partially ordered comparisons between labels must be implemented with the same semantics at each site. This view of types and operations as data is derived from an object oriented [Gold83] model of computation and it reduces the distributed TCB concern that similar things be accomplished in similar ways to yet another data consistency constraint. This data changes so slowly that an automated protocol is not usually used to guarantee consistency, but rather consistency is addressed by trusted distribution of software.

2.2 Trusted Paths between TCB Components

Within a single domain TCB, it can always be assumed that TCB local data have been defined by trusted code and that parameters passed in a procedure call have been sent by trusted code. When reasoning about the correctness of a single domain TCB, one need not question whether internal communications is being spoofed.

A distributed TCB must provide trusted paths between its distributed domains in order to achieve similar assurance. This usage of the term "trusted path" is a strengthening and generalization of its usage in [DoD85]. As used here, a trusted path offers the following guarantees:

- a. A message received from a trusted path originates from a trusted source. This property can be supported in stronger form by authentication of the exact identity and security attributes of the originating component.
- b. A message received from a trusted path contains the same value that was sent. This guarantees that message data have not been modified by untrusted entities.
- c. If messages have security labels, then the label on a received message has the same value that was sent. This guarantees that message labels have not been modified by untrusted entities.
- d. An optional property is the preservation of message order on pairwise trusted paths. (This property also prevents replay of messages.) It is optional

because it may be expensive to implement and difficult to verify. Further, it may not be required to support TCB correctness.

2.3 Trusted Protocols

A distributed TCB will need to trust some of its protocol interpreters, possibly at several different ISO levels, for any of the following reasons:

- a. To implement the trusted path concept of the previous section. Trusted path could be incorporated into the services offered by interpreters of standard protocols at the transport level and below. In the absence of such a standard, a system specific end-to-end protocol layer can be inserted at the transport level using cryptographic authentication techniques. An example of the latter approach is given later. The design verification costs for these two approaches vary considerably: verification of standard protocols is quite difficult, but given an acceptable formal model of encryption, end-to-end implementations of trusted path that do not guarantee delivery are substantially easier to verify.
- b. To implement system level atomic state transitions. If the system's security relevant integrity constraints are not very strong, this may not pose a problem. Otherwise it may be necessary to design application level protocols with the goal of taking the entire system from one consistent state to another. We will see examples of both cases later in this paper. Application level protocols defined for this purpose can be exceedingly difficult to verify.
- c. To provide system level concurrency control. Protocols such as the two phase commit can be viewed as implementing a distributed lock mechanism. For the most part, concurrency controls are useful to help in achieving security relevant atomic state transitions, but they are frequently also useful in controlling non-security relevant transitions in the system. (Recall that our definition of security does not address denial of service.)

2.4 Hierarchical Trusted Computing Bases

Distributed TCB components may be implemented as applications software on devices which also support untrusted applications, in which case a local reference monitor is required to prevent interference with trusted operations. In addition, if the distributed TCB components handle multi-level data, the local reference monitor must provide a multi-level-secure environment. So a distributed TCB may be implemented as a hierarchy of TCBs in which the system level TCB relies upon correct policy enforcement of local TCBs for its own correct policy enforcement.

The relationship between the system reference monitor and the reference monitors of individual devices can be subtle. The local TCB's interpretation of subjects and objects bears no necessary relationship to the distributed TCB's interpretation. This is particularly true if the system TCB does not view reference monitor data structures as as objects which are labelled and subject to access controls. In a distributed TCB, system level reference monitor data is likely to be application level data to the local reference monitor. Another example of the cognitive gap between local and system TCBs is that active components viewed as untrusted with respect to local policy may well be trusted with respect to the system policy, e.g. system level access control decision making and system level audit data recording.

Clearly there is a need for some "glue" to tie the various components of a distributed TCB into a consistent system level reference monitor. One of the most important such adhesives is the globally consistent representation of security labels and their comparisons. This was identified earlier as a form of integrity constraint over replicated data. Consistent labelling need not mean identical labelling, except for the external representation of labels that are exchanged over a network. Labels internal to a device may have increased granularity as long as homomorphism is maintained between internal and external forms; i.e. a well defined mapping between labels exists that preserves the dominance relation. This freedom to increase label granularity can be quite useful, particularly in the area of added compartments and subcompartments.

2.5 Fault Tolerance

Unlike a monolithic TCB, which is either in service or out of service, a distributed TCB continues to enforce a security policy when some of its components are not in service. The normal case for a distributed system is that something is broken somewhere. In consequence, a distributed TCB must support "fail-secure" properties in its design, verification, and architecture. Fail secure properties assert that no compromise occurs even when some components are unavailable.

Desirable security properties are frequently "safety" properties; i.e. properties that assert the preservation of a secure system state. As long as the failure states of the computation are secure, one need not show that a computation makes progress in order to show that it is secure. This observation allows the sidestepping of a number of known difficult verification problems, such as termination. In a distributed TCB, the granting of an access request may involve a chain of actions by different components. If the chain is broken by component failure or communication failure the result is denial of service, not compromise. End-to-end checks which do not require reasoning about the concurrent interaction of components may suffice to demonstrate fail-secureness without guaranteeing actual delivery of a service.

Although not required by the security policy presented in this paper, denial of service is of serious concern to many end users of trusted systems, in some cases of higher concern than security policy enforcement. A trusted distributed system cannot be allowed to shut down when one of its components fails. The system must continue to provide at least partial policy enforcement for as long as possible.

The classical reliability technique of replicating data and processing can be applied to distributed TCBs. As mentioned previously, one result of such replication is the necessity for synchronizing protocols to manage updates to replicated security relevant data. Another complication is that device level secure initialization and/or recovery becomes complicated by the necessity to synchronize the state of the local TCB data with the system TCB state.

3. A REAL EXAMPLE OF A DISTRIBUTED TCB

Each of the issues raised in the previous section has been addressed in the design for a classified system now in the final stages of development. For the purposes of this paper we will call this system NRM, which stands for Network Reference Monitor. The following description is greatly simplified in order to focus attention only upon the security architecture of the NRM.

NRM consists of a family of devices which, when added to a packet switched network, collectively enforce a security policy on the exchange of messages between hosts on the network. Encryption is the basic domain separation mechanism of NRM. The device types in this family consist of the following:

- a. Secure Network Interface (SNI): One of these devices interfaces each network host to the network. It is transparent because it presents a host interface to the network and a network interface to the host. This device only passes messages to or from a host after a NRM security policy check described below. A SNI encrypts a message before sending it to the network or decrypts a message delivered from the network using a key that is shared only by the source and destination hosts of the message. This key, in association with other security related data, establishes a bidirectional cryptographic connection between the two hosts. A SNI can manage a large set of cryptographic connections.
- b. Key Control Center (KCC): This device generates a new key to be used for each new cryptographic connection and securely distributes a copy of this key to the source and destination hosts associated with the connection.
- c. Security Control Center (SCC): When a host sends a message through a SNI, and the SNI does not currently manage a cryptographic connection between the

source and destination hosts of the message, the SNI requests the establishment of a new cryptographic connection by sending a request message to an SCC. The SCC mediates this request by checking for inclusion of the candidate message's security level in the security ranges of both the source and destination hosts. SCC mediation also includes a check that the message's source and destination hosts are included in each others discretionary access control lists. If both of the above checks are passed then the SCC directs the KCC to establish a cryptographic connection between the two hosts.

3.1 NRM SYSTEM SECURITY POLICY

The NRM System Security Policy has components that jointly control the establishment and use of crypto connections between pairs of hosts. For the NRM System, the range of security levels associated with each host indicates the range within which that host can communicate (i.e., send or receive messages). Each crypto connection established by the NRM System has associated with it a single security level. Thus, a connection must be established for each level at which a pair of hosts wishes to communicate.

By first controlling access of hosts to connections, and then controlling the use of those connections by the hosts, the NRM System effectively controls the flow of classified information between hosts. To accomplish this, a security policy has been defined with mandatory and discretionary components to control the access of hosts to connections, and an entelechy* component to control the use of those connections. A fourth component, the delta component, written Δ -component, limits changes to the SCC's security relevant databases.

The mandatory component of the NRM System Security Policy states that:

A host may have current access to a crypto connection only if the security level of that connection falls within the security-level range of that host.

The discretionary component of the security policy states that:

A host may have current access to a crypto connection only if that host had discretionary permission for that connection when the access was first approved.

The entelechy component of the security policy states that:

A host may send or receive messages over a crypto connection only if it has current access to that crypto connection.

* Entelechy: the realization of form-giving cause as contrasted with potential existence. (Webster's New Collegiate Dictionary)

Finally, the Δ -component states:

Only the System Security Officer can change the security relevant data in the SCC's databases.

3.2 MODEL AND CORRESPONDENCE

In order to meet the Orange Book requirements for an A1 class of certification, it is necessary to demonstrate that the design of the system, as expressed in both the Formal Top Level Specification and the Descriptive Top Level Specification, is consistent with a formal mathematical model of security. For the NRM system, the model that was chosen is the Bell-LaPadula model, modified where necessary to more precisely express the NRM security policy. This section describes the correspondence between the model and the NRM system design.

3.2.1 Subjects and Objects

The subjects in the NRM System are hosts. A host is defined to include subscriber hosts which are directly attached to SNIs, NRM control nodes (i.e., SCCs and KCCs), or the internal host within each SNI whose function is to coordinate with the control nodes.

The NRM System objects are the connections between pairs of hosts. A connection indicates the potential for two hosts to communicate with each other by sending and receiving messages via their SNIs. Each connection is uniquely identified by a host-pair and a security label, e.g.

((host1,host2),label)

3.2.2 Security Level

In the NRM system, security level is defined exactly as it is in the Bell-LaPadula model. That is, a security level is a pair

(classification, set-of-categories)

where classification is totally ordered, and categories are not ordered. Security levels are partially ordered by the 'dominates' relation.

3.2.3 Access Modes

The Bell-LaPadula model identifies four types of access which a subject may have to an object: read, append, write, and execute.* In the NRM system, sending a message via a connection is viewed as an append to the connection-object, while receiving a message via a connection is viewed as a read from the connection-object. Since all connections in NRM are two-way (send and receive), write access, which includes both 'read' and 'append' capabilities, is the only mode of access applicable to NRM connections.

* Note that 'read' means read-only, 'append' means write-only, and 'write' means read-and-write. This awkward usage has historical origins.

Consequently, write access is the only access mode implied in the NRM System's current accesses and discretionary permissions.

3.2.4 Current Access Set

Rather than being stored in one central location, the NRM System Current Access Set is distributed among the SNIs, and is implemented as connection state records stored at each SNI.

3.2.5 Access Permission Matrix

In the NRM system, discretionary permission is defined for pairs of hosts, e.g. (host1, host2)

This represents discretionary permission for host1 to have write access to any connection object which has host1 as one end point and host2 as the other endpoint. This form is equivalent to a set of discretionary permissions as represented in the Bell-LaPadula model: for a permission, (host1, host2), the equivalent entries in the Bell-LaPadula model's Access Permission Matrix would be all entries

(host1, ((host1, host2), label), write) where label is any security level defined in the system.

The Access Permission Matrix is represented in a SCC data base as a set of Access Control Lists which represent inclusion or exclusion by host name and inclusion or exclusion by named group. For a NRM system which consists of a single domain, the Access Permission Matrix is centralized, since each SCC in the domain has a complete copy of the discretionary permission databases. However, in multi-domain systems, the matrix is distributed across domains, with each domain having only those entries of concern to the hosts in that domain.

3.2.6 Level Function

The Level Function (f) of the Bell-LaPadula model is a triple of functions:

fS = Maximum security level of a subject
fC = Current security level of a subject
fO = Security level of an object

Two of these functions are meaningful in the NRM System: the fS function and the fO function. Rather than store only a maximum level for a host, and then also store an attribute which indicates if the host is trusted, the SCC data base contains a range of levels for each host. Untrusted hosts have single level ranges and trusted hosts have multi-level ranges. There is no concept of hosts changing their "current" level, so the fC function is defined to be the same as the fS function. For objects, the fO function indicates the level of a connection, i.e., the label component of a connection identifier

((host1, host2), label).

3.2.7 Security Policy

As described above, the NRM security policy has a mandatory component, a discretionary component, a Δ -component, and an entelechy component. The mandatory and

discretionary components satisfy the Bell-LaPadula security properties, while the Δ -component and the entelechy components have no direct counterparts in the Bell-LaPadula model.

Simple Security Property

Since the mandatory component requires that the connection-object label must be within the range of the host, the maximum level of the host's range (i.e., fS for that host) dominates the level of the connection-object, thus satisfying the simple security property.

*-Property

By the mandatory component, subjects which have single-level ranges can only have current access to connection-objects at that level. This is sufficiently restrictive to satisfy the Bell-LaPadula *-property. However, note that subjects which have multi-level ranges can have current access to connection-objects at any level within their range, which is a violation of *-property. In other words, multi-level hosts are trusted subjects. The NRM mandatory component is somewhat more restrictive than the *-property in that the *-property allows trusted subjects to have write access at any level dominated by the subject's level, whereas the NRM mandatory component limits such access to only those levels which are in the subject's range.

Discretionary Policy

This component of the NRM security policy is expressed in such a way that it is only at the time the access is granted by the SCC that the system ensures the existence of discretionary permission. After this point it is possible that the discretionary permission can be invalidated in the SCC, while the current access is still active (i.e., a permission still exists in the SNI's table). * Other than the revocation issue, this component of the NRM policy is identical to the Bell-LaPadula discretionary property.

Δ -Property

Although the Tranquility Principle focused solely on changes to an object's security level, a more general statement of the principle would be that changes to the security-relevant data of the TCB cannot be made except by agents which are trusted to violate tranquility. ** This is in fact what the Δ -component addresses. In the NRM system, the only agent trusted to change the security-relevant data is the System Security Officer.

* Note that this is very similar to the situation which exists in some operating systems, where a user's current access to a file is not revoked when and if the discretionary matrix entry is deleted. Instead, the user will not be aware that discretionary permission has been revoked until he tries to access the file at a later time after giving up his current access.

** Note that inclusion of such a principle as a required property of the model would address the problems pointed out by McLean in System-Z.

Entelechy

The entelechy property was added to the NRM security policy primarily because of the distributed nature of the system, because the decision-making described in the property is implemented in software, and because this decision-making is crucial to the enforcement of security in the system. In an AI operating system, the analogous mechanism would be the memory-mapping hardware, which is usually considered to be outside the scope of the Bell-LaPadula model and formal specifications thereof. In the NRM system, the enforcement of entelechy is the primary charter of the trusted software within the SNIs, and without the correct enforcement of this property, the decision-making of the SCCs would be of little, if any, value.

3.2.8 Rules of Operation

In the Bell-LaPadula model, the possible state changes of a system are described by 'rules' which correspond to specific actions which are performed by the NRM system. This section identifies and briefly describes those actions.

3.2.8.1 Modifications of the Current Access Set

In the NRM system, modifications to the current access set are accomplished by each SNI as updates to its connection table. Adding a connection to the table is equivalent to adding an access to the Current Access Set. Removing a connection from the table is equivalent to removing an access. The SNI adds connections to its table only if they have arrived via a trusted path from the SCC via the KCC. Removal of permissions is done either in response to a command received from the SCC (again, on a trusted path), or as part of an LRU replacement mechanism when the table is full.

3.2.8.2 Modifications of Subjects, Objects, and Levels

In the SCC, two of the transactions which the Security Officer may perform are Create-Site and Delete-Site. Create-Site accomplishes the addition of a subject host. The addition of a new subject to the system implicitly adds to the system all connection objects which have that subject as one of the endpoints. Conversely, the transaction Delete-Site involves the removal of subject hosts (and implicitly their associated objects) from the SCC's database. The security range of a subject is established when the subject is added to the system, and cannot be modified once it is established. The only way that a subject's range can be changed is to delete the subject, and then add the subject with a different range specified.

The level of a connection is an integral part of the identity of the connection, and all possible connections between subjects are viewed as existing as long as both subjects exist. Thus, it makes no sense to change the level of an object, and no provision is made in the system for accomplishing this.

3.2.8.3 Modifications of the Discretionary Matrix

In the SCC, transactions have been defined for adding or removing entries from a set of relations which implement discretionary Access Control Lists: Include-Host, Exclude-Host, Group, Include-Group, and Exclude-Group. From the point of view of modifying the discretionary matrix, these transactions are somewhat obscure in their results. For example, adding a host pair to the Include-Host relation will result in that host pair being added to the (abstract) discretionary matrix only if the same host pair is NOT an entry in the Exclude-Host relation. Identifying a host as a member of a group may add or delete entries from the (abstract) discretionary matrix, depending on what entries are currently present in the Include-Group and Exclude-Group relations.

3.2.9 The Basic Security Theorem and Induction Hypothesis

The formal specification methodology being used for the NRM system is the Formal Development Methodology (FDM), developed by System Development Corporation [Sch85]. The FDM specification language, Ina Jo, permits the description of a system as a state machine. The theorems generated by FDM demonstrate that the system starts in a secure state, and that each state transformation preserves security, as defined in the criteria and constraints of the specification. This is very similar to the approach described by Bell and LaPadula in their discussion of the basic security theorem. In [BLP76], Bell and LaPadula state that "...the basic security theorem establishes the 'inductive nature' of security in that it shows that the preservation of security from one state to the next guarantees total system security." (p. 20) Based on this, it can be argued that verification of the NRM specifications demonstrates that the NRM system design is secure, as defined in the Bell-LaPadula model.

3.3 FAULT TOLERANCE

The NRM system has several strategies for continuing to provide service to application components when a NRM component has failed. These strategies include load sharing, redundant security data bases, the continuation of existing service in the absence of a control center, and secure recovery of failed components.

NRM is organized into control domains which partition the SNIs of the system; i.e. each SNI belongs to a control domain and no SNI is in more than one control domain. Each control domain has several redundant SCCs and several redundant KCCs. The SCCs of a domain share the domain workload according to a static assignment of SCCs to SNIs as a primary server. In the event of an SCC failure, the SNIs which view the failed SCC as their primary server will redirect their service requests to an alternate SCC, again according to a static assignment of secondary servers.

Since domains are disjoint, minimal data base information is shared across domains (primarily the identities of other domains and their control centers). Inconsistency of this small amount of data across domains results in denial of service, not compromise. Within a domain, SCCs maintain identical data bases which must be updated concurrently. In order to assure data base consistency across SCCs, a two phase commit protocol [Gray78] is used for data base updates which synchronizes update requests. No updates are allowed unless all SCCs are available. This procedure prevents most possible causes of data base inconsistency, but is not proof against awkwardly timed failures during the execution of the data base update protocol. We assume that such failures are disabling, and that the potential inconsistency will be identified during a secure recovery procedure which compares data bases with other SCCs. A much more detailed discussion of the SCC design to assure data base consistency is in preparation.

In the case where all SCCs of a control domain have failed, the NRM system continues to serve application hosts using in place crypto connections, but no new connections can be established*. For important or frequently used connections, the SCC keeps a list for each SNI of a set of connections to be established at the time the SNI is initialized.

Perhaps more difficult than continuing to serve in a degraded configuration is the problem of recovering a failed component without disturbing the system. NRM has been designed so that each control center establishes consistency with the other control centers in its domain before beginning to honor service requests.

3.4 CONCURRENT ISSUES

A NRM domain can be viewed as having several critical regions with respect to concurrent activities in the domain. This means that the NRM system operations must not overlap each other in time when they involve the same critical region. Actually, in the interest of improving system efficiency, the NRM design allows certain race conditions in regions that are fail-secure.

The most common critical region in a domain is a crypto connection: SNIs at each end can concurrently request establishment from different SCCs. The resulting race has four cases, two of which permit communication over the connection and two of which do not. It was decided that synchronization of SNIs to prevent this race had too high a cost in network traffic and reduced domain workload capacity. Instead the broken connection is repaired in the same way as any other: by repeating the connection request.

* The actual system upon which NRM is modelled has an alternate service capability in this situation. This capability is not described here.

The more critical region is the SCC data base, in which only a single data base update is permitted at a time. A two phase commit protocol serves as a distributed lock to assure this. It would have been possible to define finer granularity regions in the SCC data base, say at a file or record level. We decided not to do this because the SCC update rate is very low and there is little to be gained from increased concurrency in the region.

Beyond the synchronization concerns of a domain, there remains the difficulty of assuring the correctness of a domain level operation that is distributed over several devices. Crypto connection establishment is the most important of these, and the NRM design relies upon control over cryptographic variables as an end-to-end check upon connection establishment in which all known failure cases are compromise free.

Revocation of existing connections is a different matter. For the reasons given in Section 2, it is not possible to verify revocation in the presence of all network failures. Instead a number of increasingly painful heuristic procedures are employed.

3.5 Fragmented TCB Domains

The NRM depends for its correctness on a consistent interpretation of security relevant types, operations, and data across all distributed components. The NRM method for assuring this consistency begins with the controlled distribution of software releases. Each software release has a cryptographically derived checksum which is checked when it is installed at a NRM site. Operational software has access to the version number of the release currently in execution, and release numbers are compared between sibling SCCs when an SCC is initialized.

3.6 LOCAL TCBS

One of the most subtle issues in the NRM design revolved around the decomposition of the system TCB into sets of trusted components that execute on different NRM processors. Each such processor needs a local TCB to provide isolation between the trusted and untrusted functions that it supports and to provide controlled sharing of data between trusted components.

One of the earliest NRM design decisions was that communications between distributed NRM components would take place over NRM crypto connections. One of the consequences of this decision is that message traffic between NRM components carries security labels just like those carried by subscriber host messages. All dialog between a SCC and a SNI is conducted at the highest level authorized for the subscriber host attached to the SNI. This convention requires that the local TCB be able to send and receive messages at multiple security levels, and to keep message data separated by level within the local processor.

The security kernel for the SCC and KCC processors has a traditional security architecture based upon a secure MULTICS model. In addition, this kernel defines and enforces an integrity policy [Biba77] which is isomorphic to a dual of the traditional compromise policy, i.e. the integrity labels are drawn from a completely disjoint set of labels. The ordered part of the integrity label is used to support internal trusted path arguments which assert that high integrity trusted components can receive data only from other high integrity trusted components. The SCC/KCC kernel does not define a discretionary policy, but the unordered integrity compartments are assigned in such a way as to create incomparable integrity domains for different TCB subsystems. This convention enforces a least privilege discipline on the application design.

3.7 TRUSTED PATH PROTOCOL

Originally, the NRM design was based on the use of TCP for transport of messages between distributed trusted components. We found TCP lacking for a number of security related reasons which are described in this section.

TCP is not a message stream, but a byte stream which may deliver bytes in different blocks than those that were sent. The NRM design adds another transport layer called Network Support Protocol (NSP) whose purpose is to block and unblock messages. NSP implements a message stream.

TCP connections are single level. The NRM design adds a security label to all outbound messages which is bound to the message text by a cryptographic checksum. Upon receipt of a message, the checksum is recomputed and, if it compares with the transmitted value, the message is assigned the transmitted label. The processing to accomplish this is organized into yet another transport layer protocol called the Trusted Path Protocol (TPP). TPP transforms NSP's single level message stream into a multi-level message stream.

The security label added to messages by TPP includes an integrity component so that a high integrity receiver of a TPP message can know with high confidence that the sender of the message was labelled high in integrity. This satisfies the source authentication requirement for trusted paths.

The cryptographic checksum applied to messages by TPP is computed using a variable which is protected in a local TCB kernel domain. This variable is shared by all NSP hosts which communicate using TPP, and is initialized and updated by trusted manual distribution. The check made upon receipt of a TPP message detects, with a high degree of confidence, unintentional or malicious modifications to message data*.

* Since TPP is at a higher level than TCP, which computes its own untrusted checksum, detection of unintentional modification should be quite rare.

4. FUTURE ISSUES

The NRM system design surfaced and dealt with a number of important issues that distinguish distributed from monolithic TCBs. There are a number of issues that were not dealt with in the NRM design, either because they do not arise in the NRM application domain, or because the NRM design sidestepped the issue. The following sections provide a brief overview of some of these issues.

4.1 Alternate Connection Models

The NRM model has been influenced by current communication protocols which rely upon positive acknowledgement and retransmission as the fundamental mechanism for assuring reliable delivery of messages. This mechanism requires data flow in both directions between the hosts involved in the exchange of a message. This is the fundamental motivation for the NRM convention that read/write is the only mode of access of a host to a crypto connection.

In anticipation of applications which use a different set of protocols, such as a trusted reliable network layer, it would be possible to define read-only and write-only access modes to a crypto connection in direct support of one-way connections.

4.2 Globally Shared Local Resources

The NRM design considers subscriber hosts to be the subjects of its policy. If a host is multi-level, it is responsible for the separation and labelling of its internal storage objects. NRM will assume that messages from such a host are correctly labelled. When a distributed system is considered in which the subjects are local subjects executing on a given host, such as a user or a process, and the objects are local resources on a possibly different host, such as a file or memory segment, a new set of issues arise. Foremost among these issues is the requirement for local TCBs at each of the distributed hosts which must coordinate policy decisions with each other. A trusted multi-level network such as NRM must be assumed to connect the local TCBs. Correct policy enforcement must rely on end to end arguments involving both the local policies and the network policy.

In this environment, a number of traditional issues become more difficult:

- a. Subject naming conventions.
- b. Object naming conventions.
- c. Identification and authentication.
- d. Audit.

4.3 Multiple TCB Interaction

In a distributed world, it is possible to view the world as a partially ordered set of abstract services, which is exactly what has been done for communication protocols in the ISO model. For each abstract service a

set of data objects and end-point entities can be defined for which it might be reasonable to define a security policy. NRM, for example, is essentially a security policy for ISO Level 3 network service (as closely as one can map IP into the ISO model). It would be absurd to define a security policy for each abstract service, but it is probably not possible to adequately address the security needs of distributed applications at the level of a single abstract service.

In the end, the security architecture of a distributed application will require both vertical integration of TCBS that are nested and rely on the policies of lower level TCBS, and horizontal integration of TCBS that interact with each other as peers in providing true end-to-end enforcement of an application level policy.

5. ACKNOWLEDGEMENTS

Unfortunately it is not possible to recognize each individual contribution to the NRM program. Over its long lifetime, NRM has been influenced by a unique team of individuals from both private and public organizations. Key contributions were made by the following. Clark Weissman of SDC has been the overall manager and technical inspiration of the NRM security team. Dan Edwards, recently of the DoDCSC, provided guidance for many of the NRM security features. The system level NRM security design was influenced by Jon Fellows, David Golber, Tom Tahan, Bob McGarity, Doug Paul, Francis Pawl, Doug Rothnie, Mark Biggar, Mary Smyrk, and Dan Faigin. The contributors to the system level NRM formal modelling and verification were Judy Hemenway, Nancy Kelem, Sandy Romero, Peter Montgomery, and Mary Smyrk.

REFERENCES

- [And72] Anderson, J. P., "Computer Security Technology Planning Study," ESD-TR-73-51, ESD/AFSC, October 1972.
- [BLP76] Bell, D. E. and LaPadula, L. J., "Secure Computer System: Unified Exposition and Multics Interpretation," ESD-TR-75-306, Mitre Corporation, March 1976.
- [Biba77] Biba, K. J., "Integrity Considerations for Secure Computer Systems," Mitre TR-3153, Mitre Corporation, April 1977.
- [DoD85] "Department of Defense Trusted Computer System Evaluation Criteria," DOD 5200.28-STD, December 1985.
- [Fra83] Fraim, L.J., "SCOMP: A Solution to the Multilevel Security Problem," IEEE Computer, July 1983.
- [Gold83] Goldberg, A. and Robson, D., "Smalltalk-80: The Language and Its Implementation," Addison Wesley, 1983.
- [Gray78] Gray, J. N., "Notes on Database Operating Systems," in "Operating Systems: an Advanced Course," edited by Bayer, R., Lecture Notes in Computer Science, Vol. 60, Springer Verlag, 1978.
- [McL87] McLean, J., "Reasoning About Security Models," Proc. IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1987.
- [Sch85] Scheid, J. and Anderson, S., "The Ina Jo Specification Language Reference Manual," TM-(L)-6021/001/01, System Development Corporation, March 1985.
- [Whit74] Whitmore, J.C. et. al., "Design for Multics Security Enhancements," ESD-TR-74-176, Honeywell Information Systems, 1974.

Specification and Verification Tools for
Secure Distributed Systems

J. Daniel Halpern
Sam Owre

Sytek, Inc.
1225 Charleston Rd.
Mountain View, CA 94043

Introduction

This paper reports on a three year project which at the time of this conference will be precisely one year old. The project is an ambitious effort in the fields of formal specification and verification, software engineering support, and security. There are two primary goals of the project. The first is to build a short term workbench to support formal specification and verification of secure distributed systems in a software engineering environment, drawing on existing tools and techniques wherever possible. The second goal is to design a long term workbench which significantly advances the state-of-the-art in providing integrated support for the design of secure distributed systems. The project is structured in three phases: studies, short-term workbench and long-term design.

Background

RADC motivation

The project is meant to fill a gap in the development of formal systems that was perceived by the RADC secure systems community in early 1986. At that time there was ongoing work devoted to the design of secure distributed databases and secure distributed operating systems but there were no projects devoted to the development of formal specification and verification tools to facilitate the building of such systems.

State of verification technology

Existing formal tools were deficient in a number of ways:

1. For the most part, the paradigms for formal specification and verification were divorced from other aspects of the software development process. HDM [1-3] is a notable exception. It tries to match the needs of software development in a number of ways. Most importantly it has a concept of hierarchical development that matches the software engineering layering approach to computer and network architecture. But

this concept was not fully developed, let alone integrated, into conventional software development processes such as testing and configuration management.

2. The limitations of existing tools was especially unacceptable in the context of the development of large distributed systems.
3. Fault tolerance and real time performance are issues which were not addressed in existing systems.

Team assembled

The project is a joint effort of four companies which bring an interesting mix of talents and experience:

- Sytek - specification and verification of secure systems such as the NASA RAP [4,5]
 - Muse tool enhancements to classical HDM [6,7]
 - mathematical talent
- ORA - experimental Ulysses verification system
 - Ada verification contributions
 - SDOS specification and verification [8]
 - extraordinary depth of mathematical talent
- CCA - distributed database work (SDD-1, MULTIBASE, LDM/DDM)
 - design support and software development tools (DDEW, PV)
- RCA-ATL - Verlangen verification system [9,10]
 - software development experience

Organization of the project by Task

The project is divided into tasks as follows:

1. Temporal properties study
2. Database consistency study
3. Fault tolerance study
4. Survey of existing tools and methodologies and exploration of enhancements
5. Short term workbench
6. Long term tools design
7. Adaptive policy specification

This work was supported by Air Force Systems Command, Rome Air Development Center (RADC) under Contract F30602-86-C-0263.

Developments to date

As of this writing, June 1987, study tasks 1, 2, and 3 are completed and work under Task 4 is in progress.

Task 1: Temporal Properties Study

Task 1 was led by Edward Schneider of ORA. Tanja de Groot and Dianne Britton of RCA AT&T Labs contributed to the study. We summarize below some of the highlights of the report, "Temporal Properties of Distributed Systems" [11].

Our model of computation consists of a collection of processes that interact only by passing messages. The only state shared between any two processes is the communication channel between them. A process is a sequence of actions consisting of a mixture of communications and internal computing. The model presumes that the communication is synchronous. A process will be described as a set of traces, where each trace is a possible behavior of the process as observed over a finite period of time. Thus a trace of a process in an environment is a finite sequence of input and output actions.

The various kinds of temporal properties have been grouped into 5 categories:

- ✦ Security
- ✦ Progress (deadlock, livelock, starvation, liveness, fairness)
- ✦ Determinism (Concurrency control and race conditions)
- ✦ Real-time performance (resource allocation and scheduling)
- ✦ Fault-tolerance (restart, recovery, reconfiguration)

Security We have developed a non-interference model of security in the context of a Rated Event System (RES). An RES has as its ingredients a set E of events, a set T of traces, a partially ordered set L of security levels, and a function lvl which maps E to L. Basically the model says that for an arbitrary trace t and level l the events in the trace of level l or less are not affected by other events in the trace.

Verification of security is complex in a system with many processes. This complexity is managed by inferring noninterference for the entire system from proofs about each of its constituent processes. In order to make this inference from constituent processes to the whole system, each process must satisfy, in addition to noninterference, two additional properties: Determinism and Universality. Determinism asserts that the output of a process is uniquely determined by the state at the time of its invocation and Universality asserts that for any state either all inputs are accepted or none are accepted.

Both the near-term and the long-term tools should be able to handle security proofs. The major requirements for these proofs is to identify the sets of events and traces for each process. The set of events should include error messages, such as the failure to meet a real-time requirement.

Any schedulers that arbitrate among non-deterministic choices must be trusted. Normally these schedulers should not receive any classified information on which to base their scheduling decisions. However the use of scheduling priorities and time-requirements in a real-time system will sometimes use such information. Such schedulers must be shown not to leak this information.

Progress We've been successful in specifying liveness in the context of Verlangen. The resulting constructs are simple and the theorems are as amenable to proof as are the theorems we've encountered in formal specification of security. Thus both the near term and long term tools can be expected to deal with this aspect of progress at the highest level of specification. Other aspects of progress such as fairness pose greater problems. We expect the specification language for the short term tools to support specification of fairness but support for verifying such properties may have to await the long term tools. Such support will probably involve enhancement of the underlying logic with temporal constructs.

Nondeterminism Requirements tend to be deterministic and a nondeterministic property can usually be transformed to a deterministic property by adding the conditions under which the property is to hold. The biggest challenge presented by nondeterminism is in specifying and verifying deterministic transactions in the nondeterministic environment of a system of concurrent processes. Mechanisms of serializability from the database world seem appropriate for dealing with this problem and we expect the paradigm and tools of the short term workbench to support these mechanisms. The long term design may go further in supporting the model of serializability as well as particular mechanisms.

To the extent that race conditions might lead to unpredictability where predictability is needed, they need to be avoided by use of appropriate concurrency control mechanisms. At the design level, it would be useful to have support for identifying potential race conditions.

Real-Time Requirements Real time requirements of distributed systems can be dealt with only minimally at the specification level. One can introduce constructs to express the time requirements. Verification that these requirements will be met can only be determined at a very low level of implementation. Thus if these requirements are

taken into account in the theory of the specification they have the effect of introducing more nondeterminism and thus negatively impacting the verification of security.

Fault-Tolerance Requirements Fault tolerance can be designed into a system. The issues that need to be considered in such a design are:

1. The failure model - the type and amount of failures that the design is to tolerate.
2. Failure detection - schemes to detect failures
3. Fault confinement - limitation of the effect of a fault
4. Verification - the correctness of the scheme, consistency with the specification model, assurance that the implementation meets the reliability requirements of the failure model.

Task 2: Database Consistency Study

This study was led by Alejandro Buchmann of CCA. Barbara Blaustein and Uspen Chakravarthy of CCA contributed to the study as did Dan Halpern and Sam Owre of Sytek. A few highlights of the report, "Database Consistency and Security" [12] follow.

The study involved interaction between security and verification at Sytek and database design at CCA. We discovered that at the specification level consistency, integrity, and security can be expressed using the specification analog of database constraints. In another respect the requirements of specifying database concerns such as serializability has led to a productive development in our adaptation of HDM. Serializability is a property involving the order of executing transactions and is thus intrinsically procedural.

Multilevel Specifications In the HDM paradigm, the place for dealing with procedural constructs is in the mappings between levels of a multilevel specification. Unfortunately, although HDM has an interesting idea of multilevel specification, the concept has not been worked out in sufficient detail to support the development of such specifications. There are two important issues in a multilevel specification where the lower level implements the upper level.

1. The procedural aspects of the implementation mappings need to be expressible in the (declarative) language of the lower level specification and
2. The presumption of atomicity as regards the upper level state-changing operations needs to be justified in light of its violation in the lower level specification.

As part of our work in this study we experimented with constructs to specify database serializability using a two level specification. We introduced the concept of a state machine trace or history to solve 1. We found that the required justification in 2 was similar to database serializability.

Consistency and Security As mentioned earlier, a unified approach to database consistency and security was established. Both can be expressed in terms of database constraints. Thus security requirements can be specified and evaluated with constraint mechanisms already available in some database management systems.

We explored various issues involved in the maintenance of consistency of a distributed database in a perilous environment and the conflicts between concerns for consistency and concerns for security. We proposed the concept of flexible evaluation of database constraints as a means of addressing both problems. We suggest three kinds of flexibility: deferred evaluation of constraints, alternative actions in response to a violation, and a more general notion of a constraint - one which allows for exceptions.

In the case of deferred evaluation of constraints some updates are allowed without consistency checking. The new data is marked as unreliable. At some later time a process checks for consistency and restores it if necessary by deleting some or all of the marked data. This approach could be useful in resolving conflict between the needs of security and consistency when consistency constraints span security levels. The potential flow of information between security levels would be avoided or reduced by deferring evaluation from update time to say the end of the day. The same mechanism could be useful in battle situations where security leaks are of secondary concern compared with real time requirements. In this case the checking of security constraints would be deferred.

Task 3 Fault Tolerance

This task was led by Douglas Weber of ORA. A few highlights of the report, "Verification of Fault Tolerance" [13], follow.

In this study we were concerned with a declarative, rather than procedural, definition of fault tolerance and what steps must be taken to prove that a system design in fact has such fault tolerant properties. Mean time to failure, a common measure of fault tolerance, was not appropriate in this context since it depends on the operating environment of the system, not the design. We dealt instead with the concept of "fault scenario." A fault scenario is a history of a system's interaction with its environment which includes not only its inputs and outputs, but also a description of failures. A system's environment will determine whether

or not a particular fault scenario occurs, usually in a random way. Therefore, a system's environment "assigns" probabilities to each fault scenario. Mean time to failure is determined by the probabilities of fault scenarios for which the system is not "tolerant."

Our treatment of fault tolerance in this study was only minimally concerned with strategies, designs, and algorithms used to implement fault tolerant systems, and only then as examples to show why a particular definition of fault-tolerance is relevant. We considered verification of fault tolerance to be a proof that a system design supports a given set of fault scenarios. We have not dealt with the problems of insuring that a system meets the requirements of its design.

Our definition of fault tolerance is similar to the noninterference definition of security. In essence it says that the system behavior in the presence of a given fault scenario is the same as the behavior in the absence of the faults of that scenario, where behavior is defined in terms of inputs and outputs.

Methods for implementing fault tolerant systems are different from the access control methods for implementing security because faults are not external events and therefore it is not possible for a system to decide immediately whether they are fault events or not.

Fault tolerance is usually implemented by redundancy. Therefore one simple way to specify fault tolerance is to specify the redundancy of state information in the design. A design is fault tolerant if it correctly maintains the redundancy as an invariant even in the presence of the specified faults.

Our approach to specifying fault tolerance involves specifying a set C of fault scenarios. With this approach it would be useful to have a way of specifying a graceful degradation property to the effect that fault scenarios only slightly worse than those specified will not reduce the system to chaos. Graceful degradation can be defined in terms of limited interference. Then we can use the same approach to specifying graceful degradation as to fault tolerance. A set of faults C' that includes the faults close to those in C is defined. An appropriate invariant for C' will result from a weakening of the invariant for C.

We experimented with modeling an example using HDM. It was possible to specify a particular redundancy design but it was also clear that more support for the concept of history or trace was called for.

Task 4 Existing Tools and Methodologies Study

This task is led by Dan Halpern of Sytek. All members of the team are contributors to the study. We report here mainly on work done at Sytek.

A Formal Specification Language for Distributed Systems For the short term tools we expect to develop a distributed system specification language (DSL) to deal with the issues which confront us: object-oriented design, concurrency, and hierarchical design. We are familiar with HDM as enhanced by the Muse tools [7] and with Verlangen [10] and these will serve as a basis for our development of DSL.

Aspects of object-oriented design and specification of concurrency have been worked out in Verlangen. We have thought about how to modify the concept of an HDM module to be compatible with Verlangen's class and process concepts. Such an evolution of HDM seems natural and nonproblematic.

We are experimenting with the HDM concept of specification levels. HDM envisages a hierarchical development where each level of the hierarchy represents a state machine. In the HDM concept a lower level machine implements the next higher level. The concept is similar to what is used in computer and network architecture. Levels are to be tied together by implementation mappings. These mappings preserve the specification constructs, i.e. types are mapped to types, state-variables to state variables, operations to operations, etc. Mappings for operations involve procedural constructs; all the other mappings are expressed in a declarative language. Typically the mapping images of the nonprocedural parts of the specification will be characterized by decreasing levels of abstraction. Theoretically, the lowest level of specification will involve types and other constructs which correspond directly to the ingredients of the target higher-order language (HOL). If the mappings are also expressible in the HOL, the multilevel specification could be converted directly into code in such a way that the layers of the specification become layers of the implementation. In practice, this perfect mapping from specification to code is unlikely for at least three reasons:

1. Restricting the specification of mappings to implementable constructs may be too constricting.
2. The specification is likely to follow the imperatives of formal specification and verification and these are not the same as the imperatives of efficient code construction.
3. Furthermore, the HDM concept that the specification is composed of levels which are complete machines seems to be unnecessarily rigid.

Nevertheless, the introduction of procedural constructs into the specification should allow the specification to get closer to the code level than it could without such constructs. Thus some implementation issues can be addressed with such specifications.

Formal Techniques and Software Engineering

We understand and subscribe to the widely held belief that the economical development of reliable software depends on a development process which pays attention to maintenance, reusability, and extendibility. We believe that object-oriented design is currently the best design paradigm for supporting these goals directly. A persuasive case is made by Bertrand Meyer [14].

We also subscribe to the belief that reusability of code implies specification reusability and in turn, that this requires formal specification of interfaces. Thus we see two somewhat different requirements for formality: those of formal verification and those of formal specification of interfaces to support evolution of software. We also believe that formal verification plays only a small part in the development of reliable systems. Certainly, given the current state of formal verification, the other parts of the development process are more important in the sense that if these are faulty the verification can be rendered useless, but if these are done well, a faulty verification will not degrade their impact. The result of these beliefs is a commitment to pay much attention to software engineering not only for the usual reasons but also as an integral adjunct of formal verification and as a process that can benefit directly from formal methods.

Therefore, in this task, besides reviewing existing verification systems such as Gypsy [15], HDM-Muse, FDM [16], and Verlangen and specification paradigms such as CSP [17] we are exploring different aspects of software engineering. ORA is looking at Ada support environments. CCA is investigating DBMS design systems. RCA is looking at configuration management.

Sytek is involved in a survey of software engineering environments. Many of the tools we have investigated concentrate on direct support for the development of code. The production of specification is only incidental to code development. We need a paradigm which emphasizes the specification as an end product and preferably one that permits a gradual hierarchical development from specification to code. Eiffel [18] developed by Interactive Software Engineering of Santa Barbara appears to be an ideal choice. It achieves the desired development goals by supporting a rich version of object oriented design and programming. Furthermore, it achieves a crucial form of flexibility in that it has a form of target language independence. The system can accommodate any programming language that can be called from C. Thus an Eiffel specification/program has an interesting

reusability feature - namely it can be reused with different degrees of completeness. If one needs or wants to use a different programming language but likes the specification and structure of an Eiffel program, it is only necessary to rewrite the low level code in the new language. The new implementation will have the same runtime and testing support from Eiffel as did the original. Thus Eiffel can be used as a PDL for Ada programs.

This kind of partial reusability appears to offer greater promise than a more rigid form of reusability. One obvious limitation on reusability of code is the multitude of programming languages. Although this proliferation is decried by some and attempts have been made to enforce a standard such as Ada, there is good reason to believe that programming languages, like natural languages, are destined to be with us in abundance. The Eiffel paradigm attempts to live with this reality and in so doing offers possibilities of more success than paradigms which assume that reality will change to accommodate them.

For us, Eiffel suggests an intriguing direction for the scenario outlined above. Our development of DSL will involve numerous design choices concerning such things as multiple inheritance, generic types, polymorphic types, and static type checking. These choices have already been resolved in the design of Eiffel. Furthermore, Eiffel contains the rudiments of formal specification and a paradigm that uses inheritance to support hierarchical development from specification to code. To the extent that we can abide by the decisions made in Eiffel, our language could eventually be incorporated into Eiffel as an enhancement. Of course, things are not likely to go so smoothly so a combined system, DSL and Eiffel, is likely to involve changes to Eiffel as well. Nevertheless, if the marriage goes well, we will have a short term workbench and perhaps a long term design far more valuable than we had a right to expect when we wrote our proposal in April of 1986.

Conclusion

Underlying this project is the belief that an environment for developing secure distributed systems which includes both formal methods and traditional software engineering can be developed. Although the belief is still far from vindicated, our initial work supports optimism in this regard. Furthermore, it appears that the attempt at this type of development in the context of addressing the needs of secure distributed systems can have a beneficial impact on the state-of-the-art in formal specification.

REFERENCES

- [1] L. Robinson, K.N. Levitt, and B.A. Silverberg. "HDM Handbook," Volumes I-III, SRI Computer Science Laboratory, June 1979.
- [2] B.A. Silverberg. "An Overview of the SRI Hierarchical Development Methodology," SRI Computer Science Laboratory, July 1980.
- [3] B.A. Silverberg, W.D. Elliot, and D.F. Hare. "Revisions to HDM and its Tools," SRI Computer Science Laboratory, October 1981.
- [4] N. Proctor, "The Restricted Access Processor, An Example of Formal Verification," IEEE 1985 Symposium on Security & Privacy, April 1985.
- [5] N. Proctor and S. Owre, "Restricted Access Processor Verification Results Report," TR-84002, Sytek Inc., July, 1985.
- [6] S. Owre and J. D. Halpern, "Muse: The Sytek Proof Processing System," TR-85007, Sytek Inc., July 1985.
- [7] J.D. Halpern, S. Owre, N. Proctor and W.F. Wilson, "Muse: A Computer Assisted Verification System," IEEE 1986 Symposium on Security & Privacy, April 1986.
- [8] BBN Laboratories and Odyssey Research Associates, "The Secure Distributed Operating System Project," BBN Laboratories Incorporated, July 1896.
- [9] D.E. Britton, "Verlangen: A verification Language for Designs of Secure Systems," Proceedings of the 8th DOD/NBS Computer Security Conference, September 1985.
- [10] RCA, "Verlangen Language Guide," RCA Aerospace and Defense Advanced Technology Laboratories, October 1986.
- [11] ORA and RCA, "Temporal Properties of Distributed Systems," Technical Report TR87003 DS, Sytek, Inc. April, 1987.
- [12] CCA, "Database Consistency and Security," Technical Report TR87004 DS, Sytek, Inc. May, 1987.
- [13] ORA, "Verification of Fault-Tolerance," Technical Report TR87002 DS, Sytek, Inc. April, 1987.
- [14] Bertrand Meyer, "Reusability: The case for Object Oriented Design," IEEE Software, March 1987.
- [15] D. Good, "Revised Report on Gypsy 2.1," Institute for Computing Science, the University of Texas at Austin, 1984.
- [16] R. Kemmerer, "FDM - A Specification and Verification Methodology," System Development Corp., November, 1980.
- [17] C.A.R. Hoare, "Communicating Sequential Processes," Prentice-Hall International, UK 1985.
- [18] B. Meyer, "Eiffel: Programming for Reusability and Extendibility," Interactive Software Engineering, Inc., January, 1987.

SPECIFICATION FOR A CANONICAL CONFIGURATION ACCOUNTING TOOL

R. Leonard Brown
Computer Security Office, M1/055
The Aerospace Corporation
P.O. Box 92957
Los Angeles, CA 90009

The *Trusted Computer System Evaluation Criteria*¹ includes the requirement that design documentation and source code of a B2 or higher class computer system be kept under configuration management during development and maintenance of the system. Furthermore, new releases of evaluated systems that are submitted to the National Computer Security Center (NCSC) for re-evaluation as the same class (maintenance of ratings evaluation) must have been kept under configuration control since the previous evaluation. As an aid to evaluation of other configuration accounting systems for use in development of a secure system, a canonical Text and Code Control System (TCCS) has been defined. This paper describes the system. This system is not intended to be built, since it is not fully defined here or in the draft guideline, nor does it have all the functionality of some existing systems. Rather, the TCCS is presented as a reference standard that a product that is under consideration for development or purchase can be compared against. The use of TCCS, or a similar tool, as an integral part of the software development cycle is described.

1. Introduction

The Aerospace Corporation has prepared a draft guideline² on configuration management for operating system software and computer hardware that describes the minimum configuration management effort required at the B2, B3 and A1 classes of the *Trusted Computer System Evaluation Criteria*¹. This guideline also recommends higher levels of effort for all systems submitted to the NCSC for evaluation. As part of the research conducted during preparation of the draft guideline, several existing automated configuration accounting systems were examined. Two were found to be in common use and also sufficient for the recommended level of configuration accounting. These were the Source Code Control System (SCCS) which runs under the Unix[®] operating system, and the VAX DEC/CMS (Code Management System)** online library system, which runs under the Digital Equipment Corporation VMS operating system. Both require the use of additional programs, *make* in the case of SCCS and VAX DEC/MMS (Module Management System) in the case of CMS.

Major features of these utilities are incorporated into the specification of TCCS. TCCS is intended as a reference standard against which one can compare prospective configuration accounting tools. If one can perform the same operations as are performed by a function in TCCS by using at most a few basic functions of the proposed system, and if the database entries contain approximately the same information that a TCCS directory and its files contain, then that system would allow an appropriate level of configuration management to be applied to development of a secure computer system.

1.1 Organization of Paper

The paper consists of the following sections. Section 2 describes the functionality of SCCS with *make*, and then that of VAX DEC/CMS with DEC/MMS. A recommended feature that neither system includes is described at the end of section 2. Section 3 has two subsections. The first describes the syntax and functionality of the basic calls of TCCS. The second gives implementation notes for the system. Again, this is not because it is intended that TCCS be implemented, but if a product is being evaluated for use with a particular development machine and its operating system, one would have to hypothesize how TCCS would be implemented on that machine and operating system in order to compare it to the product being evaluated. Section 4 describes

*Unix is a Trademark of Bell Laboratories

**VAX is a trademark of Digital Equipment Corporation

how TCCS would be used during the development of a project which was subject to the requirements of DOD-STD-2167³, *Defense System Software Development*. This does not mean that the NCSC will require that secure computer systems be developed to this government standard, but this standard is well known and is similar to the software development cycle used by many vendors. The Appendix contains the Backus Naur Form for the simple grammar of TCCS; these calls would typically invoke a particular interactive function which would then prompt the user for the information required to complete the operation. On some systems, a command processor would have to be invoked first; on other systems, the functions could be called from the top level command interpreter. The intent of including the syntax specification is to show what parts of an instance of TCCS are dependent on its implementation and which depend on the operating system.

2. Existing systems

A number of software developers have created the kind of automated document control facility that proper configuration accounting requires. Text, both from source code and from the other documents involved in the development of software and hardware, can be entered and modified only through use of the automated system, although any programmer can get a working copy of the current developmental configuration for purposes of modifying the source code or documentation, or testing the latest version of the software. Updating the source code or document must be done only by personnel with permission to make such updates. The examples discussed below are partially dependent on the discretionary access control mechanisms of their existing system, but each system records who made each update; in addition, a reason for update may be asked for. In addition to the existing systems described here, most commercial database management systems (DBMS) can be used for configuration accounting by creating a front end processor that interfaces to the query language processor of the DBMS. If a DBMS is used, then it must have only read or write access to the records, and all updates must be made through its query language.

To motivate the list of general functions given below in section 3.1, a description of two similar systems is given here. Under the Unix[™] system, the *make* utility, and the elements *admin*, *get*, *prs* and *delta* which comprise the Source Code Control System provide a basic configuration accounting system. The Module Management System (VAX DEC/MMS) and Code Management System (VAX DEC/CMS) which run under the DEC VMS operating system provides similar facilities. In fact, MMS is modeled after *make* and has an almost identical syntax.

2.1 Unix[™] SCCS

The SCCS system runs under the Unix[™] operating system. There are several good references on it, including an overview⁴ and a manual⁵. The steps of configuration accounting corresponding to the life cycle steps described in DOD-STD-2167 require a series of function calls from the operating system shell. Initially a directory is created using the *mkdir* function. At this point, it is possible to use the owner, group, world protection scheme provided by Unix[™] to protect the directory. In addition, a list of login identifiers is created to specify who may update each element to be processed by SCCS. Some protection strategies are discussed below.

For notational purposes, each entry in the directory is referred to as an element. Following directory initiation, each document is entered initially using the function *admin -n* (the *-n* modifier specifies this is a new element). As each update is made to an element, a new generation of that element is created. SCCS

calls each new generation a *delta*. Each element is stored in a file by SCCS, and the filename is prefixed by *s.*; any files added to the directory that do not meet this requirement are ignored by the SCCS function calls. A number of arguments may be specified when *admin* is called. These arguments specify parameters that affect the file, and may be changed by a subsequent call to *admin*. For example, one such parameter indicates whether branches may be created for an element. The *alogin* argument is used to create the equivalent of an access control list by listing *login* names of users who can apply the *delta* function to the element, thus creating either a new generation (delta) or a variant branch. Setting the *v* flag causes a prompt for MR (Modification Request) numbers to be issued on any update. The *admin* function is also used to change any flags or parameters.

During the initial writing of source code, the programmer keeps the code in his own directory until it will compile and pass a few simple unit tests. The initial release, or initial delta, of each code module is inserted into the SCCS directory by means of the *admin -n* function. The programmer may update each such module by using the *get -e* function which indicates that the module will be edited, and then the completed document will be reentered into the directory using the *delta* function. As long as the module being edited was extracted from the SCCS directory using *get -e*, it can be returned to the library using *delta*, and all necessary update information will be entered with it, including the MR number if *admin -fv* has been called to set the *v* flag. The *get* function can be used to extract a copy of any document, but after it is edited it cannot be reentered into the directory. *Get* is useful for printing out copies of documents, running test compilations when some other module is being modified, or to allow more than one team member to work on the same document since the project manager can then use *get -e* and *delta* to enter the final, approved changes.

When the code is to be tested, *make* can be used to generate a test build. This function looks for a file named *makefile* in the current directory and tries to create a new version of the file named on the first line. Since this is usually an executable file, it checks to see whether all the object files needed by the loader to create this executable file are up to date, which is only true if the source files are up to date. In other words, the *makefile* gives the dependencies of an executable file, and makes sure the last modified date of any file is the same or earlier than that of any file that depends on it. When such is not the case, the contents of *makefile* specifies what action to take, or if no action is listed, searches a list of default actions. For example, if *kernel.o*, an object file, must be updated because *kernel.c* is newer, then *make* will automatically run the C language compiler on *kernel.c*. If the source files are kept in the SCCS directory, then *make* must *get* the needed source files from there. A *.DEFAULT* entry in the make file can be used to apply *get* to all needed source files if any of the object files require updating.

Another concept that is useful in integrating and testing software is that of the software build. During the testing phase of software development, a subsystem of modules can be integrated into a single executable load module and tested. However, while this testing goes on some of the source files may still be under development. Testing a software build requires a stable set of files. SCCS and *make* can handle this in one of two ways: cutoff specification and branching. If no source files will be modified during the testing, even to correct minor syntactic errors, then the *makefile* that creates the build can specify on the *get* function that only deltas made by the testing start date are to be included. Thus, the same versions of the source code are always retrieved. Alternatively, if some minor debugging will be allowed during the testing, while the development team continues to work on the source code so it will interact correctly with a later test build, then each element of the source code can be split into two or more branches. One branch will only contain the minor debugging changes made by the testing team, while the other branch will contain changes made by the development team. When testing is finished, all changes made during testing must be incorporated with the current development team code.

SCCS provides the capability to specify a software build by the way it assigns an SCCS Identification number (SID) to each output of the *delta* function. Then one can *get* any version of a text or source code file by specifying the appropriate SID. The form of the identifier is R.L.[B.[S]] where each of R, L, B and S is an integer. R stands for the Release number, which is initially 1 and must be forced to increment by a specific user action. L stands for Release Level. The project manager may decide to allow several branches to be created within the same file, either with the intent of later incorporating these branches into the same document, or of having a different branch for each possible hardware configuration, or each possible subset of peripheral devices, or for some other reason. In that case, the optional B stands for the branch designator and, for each branch, the S stands for the sequence number. Straightforward rules define how to specify the particular SID desired when *get* is called. If no SID is specified then the latest release and level is provided. A branch must be explicitly named as an argument to *get* for it to be retrieved. The SID of the resulting call to *delta* is also affected by the SID used when *get -e* is called. A table showing these rules is provided in the description of the *get* function in the *Unix™ Programmer's Manual*⁵.

Two versions may be incorporated using the *get -i* list function, specifying the most recent sequence number of each branch. The user who executes this will be notified of any conflicting modifications and must handle these interactively.

The function *prs* allows for configuration audit, since it extracts information from the *s.* files in the SCCS directory and prints them. *Prs* can be used to quickly create reports which list one or two important values, such as last modified date, for many SCCS files, or many values for one or two files. Larger reports can also be created and processed using an editor.

2.2 VAX DEC/CMS and DEC/MMS

The configuration accounting system called VAX DEC/CMS⁶ is also used to track a history of each text file stored in a CMS directory, but CMS does significantly more auditing and cross checking than SCCS does. For example, if an editor is used directly to modify a file in a CMS directory, any further use of that file by CMS generates a warning message. Any files entered into a CMS directory by other than the CMS utility will cause CMS itself to issue a warning message when it is invoked for that directory. Otherwise, the process of configuration accounting is similar to that used with SCCS.

The CMS CREATE LIBRARY function causes a directory to be set up, and initial logging to start. The project manager enters each element into the directory by using the CMS CREATE ELEMENT function. One must RESERVE an element of a library to modify it, and it can be put back into the library only by using the REPLACE function. If someone else has RESERVED an element between the original programmer's RESERVE and REPLACE calls, a warning is issued to both programmers and the occurrence is logged. To get a sample copy of text, such as a program source, the FETCH function will generate the latest generation, or any specified generation, of an element, but will not allow a modified copy to be reinserted into the library. The SHOW function can be used to audit the information about each element in the library.

MMS⁷ is almost identical to *make*, even down to using the default name *makefile* if its first default description file *DESCRIP.MMS* is not in the current directory.

Differences between SCCS and DEC/CMS appear concerning software builds. In Unix™ a build must be either described in a *makefile*, or else each element to be used in a build must be retrieved from the SCCS directory using *get*, placed in another directory, and the *makefile* then may refer to these source files to create the executable build. In CMS, the process of selecting only a subset of source files, including some which are not the most current, is automated by the use of the class and group mechanisms. To see how this works, one must understand the CMS concepts of generations and variants. Each generation of a file corresponds

to a UnixTM delta. Generations are normally numbered in ascending order. CMS also has the capability of creating a variant development line to any generation by specifying in the REPLACE function a variant name. For example, if one RESERVES generation 3 of an element, then performs a REPLACE/VARIANT=T, this will create generation 3T1 which may then be developed separately from generation 3. The first time this is used, the equivalent of an SCCS branch delta is created. Branches themselves can have branches, a capability that SCCS does not have.

A group can be defined within a CMS directory, using the CMS CREATE GROUP and CMS INSERT ELEMENT functions. A group is composed of all generations, including variant generations, of all elements inserted into the group. Groups can be included within other groups. Groups can be defined with a non-empty intersection so that they have overlapping membership. The DESCRIPTION file used by MMS can specify the groupname in a CMS FETCH function on the action line of a dependency rule. This would then fetch the most recent generation of each member of the group, including all variants. This is not all that useful during development since, as was mentioned above, the most recent generation may be changed by the development team during the course of testing a build. However, once all variants are removed and the CMS library has stabilized, a CMS FETCH function on a group name might be useful.

A more interesting case is the CMS class, which consists of specified generations of some subset of elements. The CMS CREATE CLASS function, together with the CMS INSERT GENERATION function can be used to specify the exact elements of a software build, and the DESCRIPTION file can then refer to the entire class by using the /GENERATION=[classname] qualifier on either the source or action line of a dependency rule. This makes the dependency description files quite simple when using MMS with CMS since the build can be defined within the CMS directory and controlled by the program manager or quality assurance team. The *makefile* required by Unix SCCS can be much more complex when it is required to describe a software build for intermediate testing.

2.3 The Bind Concept

One thing that SCCS and DEC/CMS lack is a way to enforce that a change to one part of the library requires a change to other elements. For example, if approval is received to change the algorithm which implements a particular function, including a change in the code, more than just the code element itself must be changed. Since no change has been made in the functional requirements of the system, the top level documents need not be changed. But the code element, the Top Level Design entries, any intermediate entries involving a description of how the system functional specifications are met by software, any documents that address how the system functions internally, and especially the test documentation and test code itself must be changed. In existing software lifecycle models this requirement is met by mapping each major function of the system down to each lower level in the top down development of the system. This can be done manually, but could be incorporated into the configuration accounting system as a series of links between elements; change in one element would not only prompt for the change authorization number that required the change, but would then lead the manager or librarian who is making the changes to update every higher and lower level document, prompting for the authorization number any time the element is accessed until a response is incorporated into the element. Current system can do this only by adding comments to elements that are intended to remind the manager or librarian to make the responses.

3. A Canonical System

The TCSEC requirements and the standard texts in software configuration management^{8,9} describe the functions that an automated configuration accounting system should provide. Inspection of the two popular systems described in the previous section suggests a workable syntax for an interactive system. These features could be implemented as part of a new operating system under

development, or through macros in a DBMS running on the development system. If the development staff is considering buying a system for configuration accounting, this provides a checklist of functions to look for.

3.1 TCCS Functional Specification

The functions of the Text and Code Control System (TCCS), and their SCCS and CMS equivalents, are summarized in Table 1. A more complete description of each function is given below. The function name is in bold type, and arguments are in *italics*. No control arguments are specified, both because different systems implement these using different syntax styles, and because the basic TCCS system described here is a minimal system, sufficient for configuration accounting but with no added functionality. However, if TCCS were actually implemented, considerations of efficiency and portability might require additional arguments.

- **setup** *directory_name* - Create a directory, or its equivalent in the current operating system environment, including access control information. The initial access control should be set with only read access to the project manager, or the entire group if the operating system allows for the group concept. Only the TCCS kernel should have direct write access to the files. The creator of the directory, and those team members whose names appear on the access control list within each element, should be allowed to use the save function.
- **enter** *element_filename* - Move an existing file into the configuration accounting directory as the initial text of a document, source code, or binary data element. Initially only the program manager would have save capability to the file. The enter function should prompt for the user identification of all team members who will also have save capability. If the operating system does not have a group mechanism, then enter should also prompt for all users who have read (actually copy) access. The enter function may be reused without the *filename* argument to update this information.
- **edit** *element_specifier_filename* - Retrieve a copy of the specified version of the element, and place it in a file of the same name in the user's current directory for editing. The default when only the element name is specified is the latest version of an element. One may also specify an earlier version or a branch version. The syntax of an earlier or branch version depends on the naming convention used by the save function. If more than one editor is available on the system, this should act as a sort of preprocessor which reconstructs the desired version without any of the TCCS header material normally stored with it.
- **save** *list_of_elements* - save the file that was extracted with the edit function back into the TCCS directory. If *list_of_elements* is a single element name, then save should inform the user of the version number to be given to the new version and then prompt for any modification to that default. This is how a new branch would be initiated. If *list_of_elements* is a particular version specifier, save will determine that it does not already exist, and that it is a valid descendant of the element named in the corresponding edit function, and assign that version specifier to the new version. If two or more branches are to be merged, then all versions of element are specified in *list_of_elements*, all changes from the latest common root version are applied, and any contradictions are signaled to the user. If the list contains version specifiers that do not arise from the same element, then an error condition is signalled and no other action is taken.
- **copy** *element_specifier* - create a printable or compilable file, based on the specified version of *element*, which does not have sufficient information to be edited and reentered into the library.
- **audit** *element_list* - The project manager, or anyone with read privilege to the TCCS directory, can direct information about the elements specified to be written as a report to the terminal, a standard output device, or a file. The *element_list* can be a *buildname* or *bindname*. The audit function prompts the user for the information required and uses a default format for the output. One useful report

would be the list of elements specified in the *buildname* or *bindname*. The format can be dependent on the output device. If sent to a file, the report can be processed with a word processor or formatter. There should be a default format, easily specified by the user, which will produce a report in the same form as that produced by *generate* (see below). This will allow the system to meet the *TCSEC* requirement for an automated tool for comparing a newly generated version with a previous version of the system.

- **build *buildname*** - specify a subset of versions of elements that can then be named with a single *buildname*. The descriptive information is kept within the TCCS directory, rather than externally.
- **link *bindname*** - create a list of elements which all must be changed, or at least annotated, whenever one element is changed. In a top down, tree structured development, this is the equivalent of a subtree of the code structure. This is in contrast to a build, which is a snapshot of a subset of nodes, none of which is a descendant of any other, at a particular point in the development cycle. Also, a bind includes the corresponding subtree of the documentation tree: design documents, any related CM plan document(s), user or maintenance manual entries, test documents such as functional and acceptance test plans. It should also include the test code itself. At a minimum, one *bindname* the designates the entire development tree should be identified. Subsidiary binds could be identified for various subgroups of the development team, or for test builds, or for variant versions that are dependent on different hardware configurations.
- **generate *makefile*** - create a new load module using the precedence information in *makefile*. This file could specify a *buildname* as the source for a target executable module, indicating any versions of source files within the build that are newer than the target must be recompiled. If no *makefile* is specified, then all source language files within the TCCS directory would be used to create a default executable file, if that is possible. As a side effect, a report of which source modules were recompiled, and which library or object modules had been modified since last generation, is produced.

3.2 Implementation Details

3.2.1 File Structure The implementation of TCCS depends on the underlying operating system and its file structure. If the file system allows for creation of a subdirectory, then each TCCS database would be in its own subdirectory. If access control can be applied at the directory level, then the TCCS directory should have read permission granted to anyone who needs access to the data, but write permission denied to everyone. Files within the TCCS directory should only be modified by the TCCS functions. Some systems allow this by giving the TCCS functions special privilege, similar to superuser privilege on Unix. Other systems may not allow this, so the write permission would have to be placed on the individual files.

Since the TCCS is meant for storing both source code and documentation, the files should be able to handle all ASCII characters. Although it is intended that object modules be kept outside the TCCS directory, using the *generate* function to retrieve the source from the directory before compilation, there should still be a way to store binary data. This would allow for documentation that includes the output of graphic systems, such as files for a laser printer or output from a graphics workstation or CAD system. If the filing system does not have a feature that handles binary data other than object files, the TCCS should include this functionality. Several techniques are available for this.

What files are stored in the TCCS directory also depends on what the operating system allows. A suggested set would include exactly one file for each named element. Information on how the various generations and updates might be handled is given below. Each *buildname* would be stored in a separate file which would include the element name and specific information on which generation of which branch is to be included in that build. This would be created by the call to

build, and processed by *generate*. Since the concept of a build is new, possible implementations for the build descriptor are given below. If the operating system directory block does not contain sufficient information about files to maintain the full functionality of TCCS, then a directory data file would be maintained within the the TCCS directory. Intermediate files created while elements are being processed would also be kept within the TCCS directory, and deleted after use. Journaling, or audit, data could also be kept in a file. To minimize problems due to system crashes, whenever an element file is being processed the actual processing should be done to a temporary copy of the file, then the name changed at completion of processing. This would also allow any function to be aborted before completion.

3.2.2 Element File Contents The actual contents of each element file must allow the recreation of all versions of an element, including earlier ones. If the system is used only when a major, and approved, change need be recorded, then the inefficiency that results from this requirement is not important since TCCS would use only a small percentage of total development resources. The element file requires some kind of delimiting character to differentiate the required variable length fields. Using a single non-printing character, such as octal 001 (SOH), at the beginning of each new section rather than different characters for each section will minimize problems caused by having multiple reserved characters.

A number of fields are required. The original text, as entered by *enter*, should be delimited. A field containing a list of users who are allowed to save this file should be included unless the operating system access control is sufficient for this. For each call to save, the information required to create the new variation from a base text is required. This has been commonly implemented by specifying which lines are to be deleted, and where to add lines that have been added i.e. instructions for a line editor to change the base text to the current text. If the way that **build** numbers variations makes it obvious which was the base text, then it is assumed that this field describes changes to that text. If it is not so obvious what the base text is, which might occur if users are allowed to create names for variations, then the branch and generation that this new generation is based on should be named. When the *edit* or *copy* function is used, each delta is applied in order to the original text to create the new text, and when *save* is used the new file should be compared to the base text and a new delta field created. A checksum field can be used for data integrity.

3.2.3 Bind Specification Two methods of specifying the bind are possible: the list method and the dependency method. In the list method, a list of all documents related to a particular element is created. When the manager invokes the *link* command with a new *bindname*, it prompts for the contents of the bind. If the *bindname* already exists, it prompts for additions. An appropriate format would be

```
name1 :: name2 , name3 , name4 , name5 ;
```

where the right hand side lists all elements that might have to be changed if a change is made in the element on the left hand side.

In the dependency method, a notation similar to that of the *makefile* syntax is used to show that a change in an element may be propagated. Unlike a *makefile*, however, such propagation may occur in both directions. For notational purposes, define up to be towards the root *System Requirements Specification*, and down toward code, then sideways can be considered an element at the same level on another branch of the tree. Consider the example of a change in the algorithm described or listed in a Top Level Software design document. A change in it would propagate up to the Software Requirements Document, and down the tree to the code document. However, a change in the test code would only propagate sideways to the test documentation, and a change in the test documentation would propagate sideways to the test code, and also up to the software test plan. An appropriate format for the dependency in a bind would be

name1<->name2

name1->name3

name1<-name4

where each actual inter-element dependency requires only one entry. In the first example entry, a change in either element may require a change in the other; in the second entry a change in name1 may require a change in element name2 but not the reverse; the third entry shows that order can be reversed by using a different symbol. See figure 1 for the partial graph of a simple project involving CPCI 1.1. Table 2 shows both the list method and dependency method entries in a bind description for CPCI 1.1.

Thereafter, whenever an element of the bind is modified, all related elements are marked with the MR number and any invocation of that element will result in a message that MR number has not been incorporated into the element yet. If an element is a member of multiple binds, the person modifying the element is queried to see which ones are appropriate to activate. For example, if the MR that led to a CCB approved change is known to affect only the subsystem of which the element is a part, and a bind has been defined for that subsystem, then that bind and not the total system bind can be named.

4. Use of Automated Tools in the Software Development Cycle

DOD-STD-2167³ describes a standard reference software development cycle which a software developer should strive to emulate. This process can be assisted by the use of a tool which implements the functions of the canonical TCCS described above. The following sections describe how such a tool would be used during software development of a secure computer operating system. Document names in italics are documents specified in the standard, and described in related Data Item Descriptions. Hardware documentation and any drawings generated by CAD equipment could also be included in the configuration accounting, but for reasons of brevity, and because the example DOD-STD-2167 is a software standard, their use is not described here.

4.1 Requirements Development Phase

The system project manager initially sets up accounts for each programmer or analyst involved in the project, and each programmer or analyst is given access to a directory of text files, an editor and word processor/formatter. Sharing of work among team members should be easy to accomplish. During requirements development, each team member writes specific sections of the requirements document, following the format shown in the *Standards and Procedures Manual*. If such a manual does not already exist, a draft version of it should be written by a small committee of experienced team members. If it addresses only code and not the format of other documents, then the team leader should develop a format consistent with the way the configuration accounting tool will store the eventual text files. The *Standards and Procedures Manual* is the only document that need not be entered into the TCCS database, although having an online copy, complete with blank format examples, is desirable.

For B3 and A1 systems, the formal security policy model and the consistency proof are among the requirements documents generated at this phase. If an existing model is being used, then this can be replaced by a reference to the existing description.

Once the project manager sees that each section of the document has been completed by the assigned analyst, although still subject to change by the manager or as the result of interaction with other team members, the manager uses the `setup` function to create a database. Each element of the requirements documentation is placed in the database using the `enter` function. Documents that depend on one another should be represented as a *bind* using the `link` function. Once this database has been created, the manager and team revise the documents,

Figure 1 - Sample Document Dependency Graph

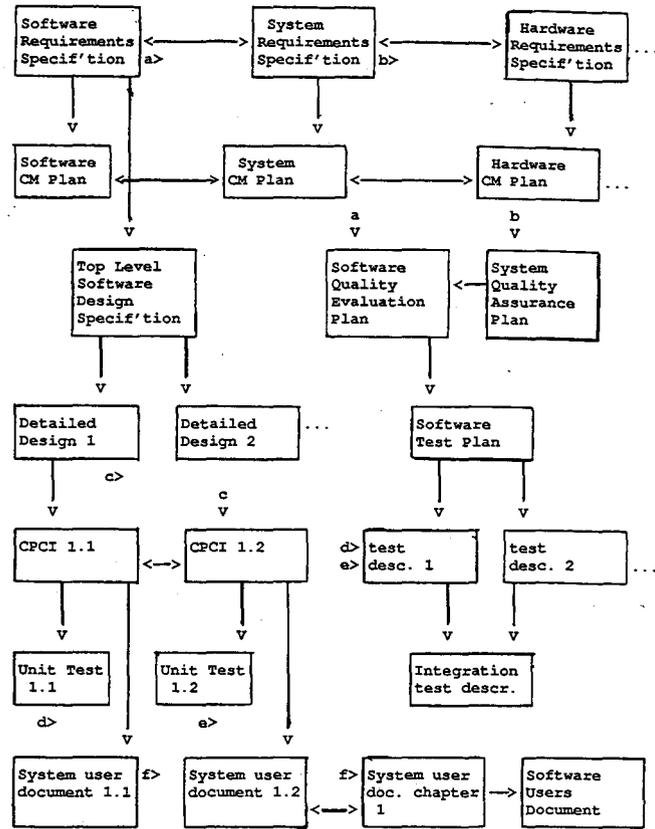


Table 1 - Equivalent TCCS, CMS, and SCCS functions

TCCS	CMS	SCCS
setup	CREATE LIBRARY	mkdir
enter	CREATE ELEMENT	admin -n
edit	RESERVE	get -e
save	REPLACE	delta
copy	FETCH	get
audit	CMS SHOW	prs
build	CREATE CLASS	delta -rsid
generate	MMS	make
link	N/A	n/a

Table 2 - Bind Specification for CPCI 1.1 in a Simple System

List Method:

CPCI 1.1 :: CPCI 1.2, Unit Test 1.1
 CPCI 1.2 :: CPCI 1.1, Unit Test 1.2
 Unit Test 1.1 :: Test Description 1
 Unit Test 1.2 :: Test Description 1
 Detailed Design 1 :: CPCI 1.1, CPCI 1.2

Dependency Method:

CPCI 1.1 <-> CPCI 1.2
 CPCI 1.1 <- Detailed Design 1
 CPCI 1.1 -> Unit Test 1.1

possibly by letting other team members revise portions of each document. The process involved in doing this is straightforward. The manager allows all team members who will revise a section to have read access to the appropriate elements. Then the team member uses **copy** to get a copy of the element, an editor to do the rewrite, and gets approval of the manager to reinsert the document. The project manager uses **edit** to retrieve the document, uses the editor to replace the changed sections with the approved files from his directory, then saves the modified document. During the **edit/save** operation the database is locked so that no one else can execute an **edit** function on that element. If every team member were allowed to use **edit** then each document would have to be broken into many smaller documents so that several team members could each work on one section at a time. This is feasible for source code, but not desirable for documentation. When the new section is saved, the automated tool automatically notes what changes were made and who made the changes.

Once the documents are ready, the Configuration Control Board (CCB) reviews the requirements documents. Any changes they require can be entered by the team manager by using **edit** and **save**, giving the minutes of the CCB meeting as the reason for the change.

4.2 Functional Specification

At the next phase of development, several documents are created. In each case, the same procedure may be used for text documents as was used for the requirements document. Thus, new database elements are placed in the system for the *Software Requirements Specification*, the *Interface Requirements Specification*, the *Software Configuration Management Plan*, and the *Software Quality Evaluation Plan*. At the A1 level, the *Verification Plan* is included. Each element should be linked to its appropriate *binds* by the project manager. In each case, **enter**, **copy**, **edit**, and **save** are used as above. Every use of **save** prompts for a MR number if the document has previously been approved by the CCB.

4.3 Developmental Phase

During the developmental phase, the modules identified during the functional specification phase are filled out, first with either graphical representations of the algorithms to be used eg. flow charts, or textual representations such as Programming Design Language (PDL). In the case of textual representations, the same techniques may be used as for other documents. For graphical representations, especially those produced on a separate device such as a CAD workstation, the **copy**, **edit**, and **save** functions can be used over a communications line connecting the workstation with the main computer. If such a communications line is not feasible, then some kind of common medium such as a floppy disk or tape will be used. In either case, the graphical representations may still be kept under configuration control by the automated tool.

When coding starts, Configuration Identification comes into play with the naming and numbering of modules. This can easily be enforced by the project manager using the **enter** function. Each new element should also be linked into its appropriate binds. Test modules are also generated for the simple tests used by the programmers to unit test these modules. A typical sequence of interactions with the TCCS database would proceed as follows. The manager **enters** a code segment, giving the first line of the module including the module name and calling sequence as described in the interface document. The manager also **enters** a blank unit test module. He **links** them together, and links the code module to its description. The programmer creates a **copy** of the document, fills out the code with reference to a **copy** of the flow chart or PDL representation of the module. He writes a unit test procedure that calls the module. He compiles both pieces of code, executes the simple test, and continues the familiar debugging cycle. Once the code passes the programmer's unit tests, the project manager calls **edit** to place the initial version of the module and unit test code into the database, or else gives the programmer temporary permission to perform the same operation.

Once sufficient code has been generated, the test plan included in the *Software*

Quality Evaluation Plan will include some intermediate testable builds. The Quality Assurance team members create *makefiles* that describe these builds, and write the required tests using a procedure similar to that described above. The **generate** function is used to create these intermediate builds. The **build** function can be used to simplify the makefiles by creating buildnames for the test builds. Once the whole build compiles and loads, the tests are run and any errors or inconsistencies are noted.

Erroneous test results require debugging and modification. Since all the modules are under configuration control, debugging is not as simple at this stage as at earlier stages. Each programmer must make sure that any changes made do not affect other modules. Again, **copy**, **edit** and **save** are used to reprogram, unit test, and replace modules. Once a build has successfully met its tests, the CCB meets and approves all modules involved. Any further changes to a module requires CCB approval and a MR number as justification.

4.4 System Integration and Testing

Once all intermediate builds are finished, the entire system may be tested. If the system is to run on the development system, this is fairly easy. It is slightly more difficult if the system is being cross compiled to another computer. The Quality Assurance team uses **build** or **generate** to create a test system, including compiling the test routines. The tests are run, any anomalies are noted, and the reports are sent back to the manager for disposition. This should be the first time that requirements and functional specifications are considered for modification. Some major requirement, such as timing or capacity, may not be met by the system. In such a case, either the requirement must be loosened, or a major redesign may be required for the system. Any change to requirements, design or code must be approved by the CCB. Any change to requirements or specification must be propagated through the design and code; any change to the specification or design must be propagated through the code. The TCCS makes this propagation easy since requirements can be traced up and down the chain of documentation by cross references to other documents and code segments within each database element.

4.5 Production Phase

Once the final build passes all tests, and after the NCSC team completes testing and approves the design and implementation the configuration accounting database is archived for reference purposes. A clean copy, without any historical data, is made of all relevant documents. All design documents, such as flowcharts and PDL descriptions, and all source code modules are also copied in a form stripped of all historical data. **Generate** is used to produce production copies of the system from this. However, Configuration Control does not end here; it continues from this checkpoint. The clean copies of all code and documentation are stored in a new database kept by the configuration accounting tool, and during the maintenance phase any changes to code, specifications, design, or possibly even requirements, if approved by the CCB, are entered into the database using **edit**. The **link** function is used to describe the relationships between the user manuals and maintenance manuals, the operational test suite, and the code source.

The functions **audit** and **generate** can be used to provide the facility to ascertain that only intended changes were made to the system version being produced. The **audit** function can be invoked to list all elements that have been added to the code since the last version. All of these changes must have been controlled by the CCB and only **entered** by appropriate personnel. Then when **generate** is used to create an object tape of the system, it will create a report of which source modules had to be recompiled due to changes since the last version. A comparison of these reports would show any discrepancies if either the makefile had been tampered with, or an unauthorized change had been made to a source file after circumventing the TCCS system.

References

1. National Computer Security Center, "Trusted Computer System Evaluation Criteria", Tech. report DoD 5200.28-STD, United States Department of Defense, December 1985.
2. R. L. Brown, "Configuration Management for Development of a Secure Computer System", Tech. report in preparation, The Aerospace Corporation, 1986.
3. Department of Defense, "Defense System Software Development", Tech. report DOD-STD-2167, United States Department of Defense, 1984.
4. Kaare Christian, *The Unix Operating System*, John Wiley & Sons, 1983.
5. Computer Science Department, "Unix Programmer's Manual", Tech. report 4.2 bsd virtual VAX-11 version, University of California at Berkeley, 1983.
6. Digital Equipment Corporation, "VAX DEC/CMS Reference Manual", Tech. report AA-L372B-TE, Digital Equipment Corporation, 1984.
7. Digital Equipment Corporation, "VAX DEC/MMS User's Guide", Tech. report AA-P119B-TE, Digital Equipment Corporation, 1984.
8. Bersoff, Edward H, Vilas D. Henderson, Stanley G. Siegel, *Software Configuration Management*, Prentice-Hall, Inc., 1980.
9. J. K. Buckle, *Software Configuration Management*, MacMillan Press Ltd., 1982.

<copy invocation>	::=	copy <element specifier>
<audit invocation>	::=	audit <element list>
<build invocation>	::=	build <buildname>
<buildname>	::=	<name>
<generate invocation>	::=	generate <makefile>
<makefile>	::=	<filename>
<link invocation>	::=	link <bindname>
<bindname>	::=	<name>

Implementation Dependent Identifiers

<directory name> is a single identifier that satisfies the local operating system syntax for naming a common group of files. In a tree style directory filing system, this would be the name of a subdirectory. In a flat file system, this would be the common prefix that all files in the group use.

<filename> is a single identifier capable of specifying a contiguous text or binary file. In a tree structured directory filing system, this would be a (possibly abbreviated) pathname. In a flat filing system, it would be the full name of the file unless the operating system allowed part of the prefix to be assumed.

<name> is a default text string that the operating system command processor would recognize as a valid argument to a function call. The name should be passed intact, as a text string identifying the TCCS element or buildname to be processed.

<element specifier> is a valid name, as above, plus whatever additional text is required to specify a particular release or level of a main or side branch of an element.

APPENDIX - Backus-Naur Syntax of TCCS

Implementation Independent Definitions

<session>	::=	<session body> end
<session body>	::=	<empty> <session body> <function invocation>
<function invocation>	::=	<setup invocation> <enter invocation> <edit invocation> <save invocation> <copy invocation> <audit invocation> <build invocation> <generate invocation> <link invocation>
<setup invocation>	::=	setup <directory name>
<enter invocation>	::=	enter <element> [<filename>]
<edit invocation>	::=	edit <element specifier> [<filename>]
<save invocation>	::=	save <list of elements>
<element>	::=	<name>
<element list>	::=	<list of elements> <buildname> <bindname>
<list of elements>	::=	<element specifier> <list of elements> <element specifier>

RACF IMPLEMENTATION AT PUGET POWER

Arturo Maria, PhD

Information Systems Consultant

Abstract

This document describes the approach taken at Puget Sound Power and Light Company to implement IBM's Resource Access Control Facility.

Introduction

During the past ten years, a very significant shift of focus has occurred in the information processing industry. This shift of focus has emphasized not only output and information deliverables, but internal controls as well. Several forces have contributed to this shift of focus including federal statutory requirements, state and local government regulations, accounting and audit firms interpretations of data security/internal control regulations and the micro-computer revolution which has brought tremendous computer power at a relatively low cost -- power which can be used for legitimate and illegitimate purposes.

To illustrate this shift of focus, the impact of the Foreign Corrupt Practices Act (or FCPA) should be highlighted. This important Federal legislation enacted in the late 1970's attempted to bring accountability for un-ethical business practices to corporate directors and officers. However, amendments to this act and interpretations by legal and auditing national firms extended this act to cover not only un-ethical business practices but crimes involving computer resources when these resources are not properly protected. Thus, the importance of internal computer controls was brought out of the technical realms and into the boardroom where it became a legitimate business concern -- the proper place for this issue.

Data security and internal controls have become corporate business problems and not technical problems. Thus, our im-

plementation of information resources controls had to be addressed first at a corporate level and secondarily at a technical level.

In our company, the need for a security package was highlighted by our external and internal auditors who commented on the need to install security packages and improve controls. This need was further highlighted by federal and state legislation such as the Privacy Act and the Washington Computer Trasspass laws (RCW 9A.52.110) which further define Corporate responsibility and computer crimes.

These combined factors and cost/benefit opportunities prompted management to authorize the purchase of a data security package and the creation of an Information Systems Access Administration group to manage the implementation of the package and the daily management of access requests and profiles.

This document describes the approach taken at Puget in installing our data security package and the problems/solutions associated with such an implementation. A special note of appreciation is extended to our Manager of Information Systems Quality Assurance, Jim Hall, for his support during the early stages of this project. In addition, Roger Deitz of Technical Support significantly and enthusiastically contributed to the success of this project by developing/installing systems interfaces and providing valuable input where philosophical decisions were required.

Defining Corporate Policy and Procedures

Since internal controls and data security issues are management issues, it was imperative that our corporate management clearly stated official *corporate*

policy on these issues. Our corporate policy on these issues is stated in our *Corporate Policy Guide Section 34 "Information/Data Security"* which states:

"All employees are responsible for protecting, utilizing, and releasing information resources of the Company in a manner consistent with the direction and standards set by the Internal Control Review Committee".

In addition, CPG-34 further clarifies the intent of this policy by stating that:

"Information Resources within any company organization are property of the company. The Company, through its employees, has a responsibility to balance the requirements for information with the need to secure its information resources from the threat of willful or accidental destruction, modification and unauthorized disclosure. Responsibility for the security of information rests with the individuals having possession or knowledge of the information".

A major/key concept in this policy section is the definition of **Information Resource Administrators (IRAs)** who are directors and managers who have authorized the creation and maintenance of corporate data. These IRAs determine who can and can not access their data. Therefore, Corporate Information Systems became a custodian (and not owner/administrator) of the data. If a corporate employee needs access to a specific resource, Access Administration coordinates the signature approval process and forwards these requests to the proper IRAs who subsequently approve and/or deny these requests. Thus, Corporate Information Systems became a *coordinator of access* and not a decision maker.

It should be noted that improper handling and/or disclosure of information is subject to disciplinary action as outlined in our Corporate Policy Guide section 11 Ethics.

Access Request Procedures

Procedures delineating steps required to request/grant access are described in our Standards and Procedures manual: *Information Systems Guide section 102.*

ISG-102 was created in order to document procedures to be followed when requesting access to online systems -- i.e. TSO, ROSCOE, CICS, VM/CMS, Model 204, etc. -- or other Information Systems resources under the custodianship of the Corporate Information Systems department.

As discussed in CPG-34, the Company, through its employees, has a responsibility to balance the requirements for information with the need to secure its information resources from the threat of willful or accidental destruction, modification and unauthorized disclosures. Central access controls are therefore required to secure these information resources and protect them.

It should be noted that responsibility for the security of information rests with the individuals having possession or knowledge of the information. Therefore, access control services are provided in order to minimize improper handling or disclosure of information.

Selection of a Package

A technical task force was formed in the Summer of 1985 to evaluate access control software systems generally available in the market. This task force was composed of Quality Assurance, Data Administration, Computer Operations,

Financial Systems, Applications Development, Customer Services, Client Services, Internal Audit and Technical Services. The mission of this task force was to develop a criteria which would be used to evaluate access control software packages that are currently available that would run in the VM environment and the MVS/XA environment.

The overall strategy called for each resource manager/technical task force member to provide a set of issues and requirements for their specific area of responsibility. These requirements would then be used in order to evaluate access control packages and make a recommendation to management.

The task force defined the original criteria for selection based on requirements that the selected software package must have sufficient market share in order to ensure that the vendor selected would be able to support a multiple vendor software environment. In addition, the selected software package must run in the MVS/XA environment and in the VM/SP environment in order to ensure protection for our overall environment, restrict access to systems interfaces and ensure that physical modifications to shared resources would be recognized across operating environments.

Based on these initial requirements, the task force reduced the number of packages to three. They were ACF2, RACF and TOPSECRET.

During the task force evaluation sessions, it was noted that at the time TOPSECRET was running in the VM environment only in selected beta sites environments and that the VM component was not scheduled for general release availability until late 1985/early 1986. In addition, our external auditors suggested that our environment should not consider access control software packages that are running in production environments for less than one year.

Based on this criteria, the packages under consideration were reduced to two: ACF2 and RACF.

The task force developed a set of requirements which were used to evaluate both packages. It was found that both ACF2 and RACF met all the technical and end-user requirements formulated by the task force. Several changes had occurred in the last 18 months (1984/1985) which made RACF and ACF2 very similar in ease-of-use characteristics and flexibility.

In addition, IBM had changed the source code distribution policy for certain products under MVS/XA where the source for operating systems modules and other selected products is not released. Since ACF2 relies on non-standard interfaces and front-end modules, several ACF2 users speculated that this change in IBM policy had caused a problem for current and upcoming releases. In addition, some ACF2 customers and other industry specialists implied that products such as ACF2 are in a period of transition since most of their interfaces to the operating system would have to be rewritten and/or modified in order to accommodate changes in MVS/XA.

With Release 1.7 of RACF, the differences in product philosophy and capabilities are almost non-existent. This was not the case in previous releases. The net effect is that today, there is very little difference between the products. In addition, IBM declared RACF as a strategic product and as such, an integration of the data management, operating system capabilities and access control facilities, we felt, is inevitable.

Therefore, the technical task force unanimously recommended that RACF should be selected as our access control software package. A summary of requirements developed by the task force are included in the document entitled "*Access Control Software Technical*

Evaluation" (August 22, 1985) available upon request.

SYSTEMS IMPLEMENTATION

SYSTEMS SOFTWARE INSTALLATION

Installation of the Operating Systems components for both environments (VM/SP and MVS/XA) is, in general terms, a 'by-the-book' process. Essentially, IBM's SMP procedures and VM installation procedures are followed and the components are properly installed.

It should be noted that we we elected not to implement a shared database environment, since the VM/SP and MVS/XA environments are not totally integrated. In addition, we felt that the risks associated with installing a shared environment outweighed the possible benefits associated with such an implementation.

USER IDENTIFICATION

Once the operating system components are installed, the next logical step is to identify all system users. This procedure necessitates a common method of identification. We elected to use a Zxxxxx standard where xxxxx = employee number. By using the employee number, RACF identification would remain constant even if the employee changed names, working locations or job titles. By being unique, the employee number also provided an excellent identifier that could be used in tandem with Human Resources supporting systems.

Once this standard was adopted, non-standard userids had to be converted to meet the Zxxxxx syntax. The first set of

systems users to be defined were our TSO clients. TSO users were chosen because the vast majority of them are data processing professionals and the TSO environment is supported by RACF with minimal modifications. This implementation was conducted in a smooth and professional manner.

The second online system to be implemented was the ROSCOE development environment. Since ADR (ROSCOE's supplier) does not provide a RACF interface, one had to be developed internally to RACHECK userids requesting access. This *ROSCOE signon interface* had to be designed/implemented so it would work with additional ROSCOE interfaces which would perform submission of jobs and data-set-access validation services.

Again, the implementation of this major development sub-system was conducted in a very smooth and professional manner.

In conjunction with the implementation of the ROSCOE/RACF environment, a forced-signon policy was enacted. In essence, our Company uses the NETWORK DIRECTOR product from Northridge Software which manages VTAM network access. This product provides the capability to require a user to logon to the system prior to being presented an applications menu which contains the major subsystems to be selected -- i.e. ROSCOE, TSO, etc.

The NETWORK DIRECTOR performs RACF validation and properly interfaces with all other online subsystems. By requiring users to logon to the systems prior to being able to make a subsystem selection, Access Administration achieved a single systems image view of online access requests -- a good place to be.

The last two major subsystems provided interesting challenges. IBM's CICS has a RACF interface. However, since the

NETWORK DIRECTOR performs RACF validation for all CICS users, CICS essentially relies on this validation if a signon-table defined user requests access.

Initially, Model 204 did not have a RACF interface. Therefore, Model 204 users were required to change their Model 204 passwords when a RACF password change was performed. We expect this requirement to be eliminated with the installation of the CCA/RACF interface by Fall 1987.

SYSTEMS INTERFACES AND EXITS

Once all online system users are identified, then all batch jobs can/should be properly identified. TSO users are automatically supplied appropriate userid/password parameters at submit time via address-space authorization propagation facilities in Release 1.7. However, a need existed to insert these parameters in the job card of ROSCOE submitted jobs. This insertion was performed by a locally developed interface which provides identification/notification services for ROSCOE users submitting jobs.

In essence, all jobs submitted to the system via TSO and/or ROSCOE are properly identified with a valid RACF userid and password. However, this was not true of production jobs submitted by Computer Operations/Operations Support. An interface had to be locally developed which would supply valid production userids and passwords to production jobs submitted. This interface was locally developed and supplies these parameters to production jobs based on a *batch control interface table* which has these parameters encrypted. Since these parameters are supplied dynamically and available to the system only, the need is satisfied and at the same time the parameters are not universally readable.

An additional interface was developed which provides RACF validation services to ROSCOE users importing/exporting datasets. In essence, all dataset access requests are properly RACHECKed prior to being performed.

Other system interfaces, such as the DMS/OS RACF interface, were purchased/installed in order to provide appropriate backup/recovery validation.

PROTECTION OF OPERATING SYSTEM RESOURCES

Operating System libraries (SYS1, SYS2, PUG) were subsequently RACF defined and protected. These libraries are universally readable but a RACF exception occurs when someone outside of Tech Support attempts to update these resources.

As part of this implementation phase, our *Change Control environment* and *Operations Support environment* were subsequently RACF protected in order to prevent unauthorized access to these resources.

PROTECTION OF PRODUCTION APPLICATION SYSTEMS

Protection of the Personnel System

Our Personnel System was the first application system whose access was controlled in background and foreground mode by our RACF profiles. As such, this system served as a pilot project which applied access control policy concepts contained in our then recently approved *CPG-34 Information/Data Security*.

In order to facilitate the protection of these resources and minimize impact, RACF profiles were defined in *WARN*

mode and monitored for approximately one month. During this period, access attempts were reviewed and researched. Valid access requests were subsequently granted and RACF profiles were modified. Subsequently, these profiles were implemented in *FAIL* mode which denied access to entities not defined in the profiles access list. The result of this implementation was a protected environment for the Personnel System (SPER) which allowed Human Resources Department personnel appropriate access to systems resources while at the same time excluding non-authorized users. This access management structure protects our company from unauthorized penetrations and willful and/or accidental destruction of sensitive corporate records.

In accordance with CPG-34, Vice President Human Resources, acting as the Information Resource Administrator for this system, was the approving entity for access definition profiles.

Protection of the Payroll System

Our Payroll System was the second application system whose access was controlled in background and foreground mode by our RACF controls. As such, this system incorporated policy concepts contained in our then recently approved CPG-34 Information/Data Security.

Again, in order to facilitate the protection of these resources and minimize impact, RACF profiles were defined in *WARN* mode and monitored for approximately one month. During this period, access attempts were reviewed and researched. Valid access requests were subsequently granted and RACF profiles were modified. Subsequently, these profiles were implemented in *FAIL* mode which denied access to entities not defined in the profiles access list. The result of this implementation was a protected environment for the

Payroll System which allows Accounting/Payroll Department personnel appropriate access to systems resources while at the same time excluding non-authorized users. This access management structure protects the Company from unauthorized penetrations and willful and/or accidental destruction of sensitive corporate records.

In accordance with CPG-34, Manager General Accounting, acting as the Information Resource Administrator for this system, was the approving entity for access definition profiles.

Protection of the Customer Environment

Our Customer Services System (CSSR) supports the Customer Services and Accounting Departments. Information provided by this system is vital to the business functions of our company. Ensuring the integrity and control of this data therefore is vital to day-to-day Company operations.

In order to facilitate the protection of customer services resources and minimize impact, RACF profiles were defined in *WARN* mode and monitored for approximately one month. During this period, access attempts were reviewed and researched. Valid access requests were subsequently granted and RACF profiles were modified. Subsequently, these profiles were implemented in *FAIL* mode which denied access to entities not defined in the profiles access list. The result of this implementation was a protected environment for the Customer Services and Accounting departments.

In accordance with CPG-34, Director Customer Services and Manager General Accounting acting as the Information Resource Administrators for these systems, were the approving entity for access definition rules.

GENERIC PROFILES

All RACF profiles defined during this implementation were *GENERIC* in order to minimize resource consumption.

EXCEPTION REPORTING SYSTEMS

Our *Exception Reporting System (ERS)* reports violations and access to RACF defined resources. These reports are essential to Information Systems Access Administration since they constitute our primary source of access information and provide information on the effectiveness of RACF protection of Corporate Information Systems resources. Daily Exception Reporting System (ERS) runstreams execute SAS programs which scan SMF Data produced in the MVS/XA system. These programs report RACF SMF records activity and summarize information provided by the RACF Report Write (RACFRW). The consolidated exception reports detail access exceptions and security violations which are reviewed daily and microfiched.

Daily Exception Reporting System (ERS) runstream in the VM/SP environment perform a similar function by scanning VM data produced by the VM/SP accounting system, and producing reports which highlight VM/SP records activity. These reports are reviewed daily and microfiched.

Our weekly synchronization runstream read the M204 personnel database and compare its contents with the RACF database profiles. Differences are reported for further action by Information Systems Access Administration.

Profile Synchronization

Procedures used to request/grant access to Information Resources (see CPG-34) require maintenance and update to a

variety of systems. Most, but not all of these systems, are managed by our Access Control Facility (IBM'S RACF). However, a repository of information indicating all access -- including non-RACF such as Walker Interactive access, M204 access etc. -- is required. As a result, a *RACF Users Database System (RUDS)* was developed. RUDS is a SAS/FULL SCREEN PRODUCT application which is used by Information Systems Access Administrator to track levels of access granted to all RACF defined users. Levels of access granted include CICS security keys, ROSCOE, TSO, and M204 capabilities and other access authorities. RUDS records are updated by the RACF Administrator as access authorization forms are processed. The main objective of RUDS is *to have a central repository that can be used as a reference point by Information Systems Access Administration and Internal Audit.*

Summary

We believe the implementation of our data security package (RACF) has been a success because it addressed access controls at a corporate level first and secondarily at a technical level. As a result, we have today a Corporate Policy on Information/Data Security, request procedures, and technical capabilities which direct and control the company's information resources and manage access requests and profiles modifications. These achievements are highlighted by RACF protected systems and application environments, such as Payroll, Human Resources, and Customer Services where appropriate access to systems resources is granted to legitimate users while at the same time excluding non-authorized users. This access management structure protects our company from unauthorized penetrations and willful and/or accidental destruction of sensitive corporate records.

MANAGEMENT ACTIONS FOR IMPROVING DoD COMPUTER SECURITY

William Neugent
The MITRE Corporation
HQ USAREUR, ODCSOPS
APO New York 09063
Tel. 011-49-6221-372710

Abstract: More attention should be focused on current computer security practice in the field. Environmental factors underlying current practice are (1) a dedicated mode philosophy and (2) occasional ineffective use of computer security safeguards. To offset these factors, improvements are needed in both training and field support. Technological improvements can be harmful if they result in a false sense of security.

INTRODUCTION*

During the last five years the focus of attention in Department of Defense (DoD) computer security activities has been on redefining and expanding policy and on encouraging the advancement of technology. This is important work, and it has greatly improved understanding of computer security. The resulting improved foundation of policy and technology should lead to continued improvement of our computer security defenses.

Now that we have a better understanding of where we want to be, it is time to look more closely at where we are. Only by doing so can we best plot our course. Furthermore, the bottom line is not policy or technology, but practice. In order to better improve current practice, we must first scrutinize it.

REPORT FROM THE FIELD

A common perception in the DoD is that system high operation is the normal current security mode of operation. Much attention is now focused on ways to advance beyond system high operation into multilevel secure (MLS) operation. From the field, however, comes a different view. In fact, the vast majority of systems still operate in the dedicated security mode of operation. This does not necessarily reflect negatively on DoD security. Rather, it is an important aspect of DoD computer security that needs to be understood.

A second aspect of current practice is occasional ineffective use of computer security safeguards. While there has been no recent definitive review of DoD computer security practice, there are cases of systems in which:

- Group passwords are used.
- Passwords are not often changed and are not well protected.
- Audit trails are not checked or even kept.
- File protection features are used haphazardly or not at all.
- Copy commands are trusted to copy unclassified files from classified disks to unclassified disks.

Some systems are operating without having been accredited. One report addressing computer use by Defense contractors lists operation without accreditation as the most common deficiency, and notes that the finding is probably equally applicable to government computers¹. Some systems have not been certified by any systematic process, other than the implicit certification that comes from operational use.

Occasional cases are not sufficient cause for alarm, and do not imply inadequate protection of classified information. Nevertheless, prudence suggests the need for closer examination of the current situation.

It is important to recognize that MLS technology does not address these matters. Indeed, the insertion of MLS technology into this environment could create a problem. In a dedicated mode system, whatever human errors are made, the person who ends up seeing the data is still fully cleared and authorized. This is true because, by definition, all users of a dedicated mode system must be cleared and authorized for all data in the system. In an MLS system, often there is no longer such protection against human error. If a user accidentally labels Secret data as unclassified, unclassified users might be able to access the data. Furthermore, some users tend to view MLS products as magic, plug-in solutions. They are sometimes surprised to learn that these products need to be adapted for their specific applications. If users do this adaptation themselves, it is possible that in doing so they will unwittingly subvert some of the protective features of the product.

Technology alone cannot be relied on to satisfy the security needs of most DoD users. Technology is merely one element in a set of safeguards, of which the most important element continues to be user

*This paper is derived from work performed under contract F19628-86-C-0001 for the United States Army, Europe (USAREUR), Office of the Deputy Chief of Staff, Operations (ODCSOPS).

practice. Before technological safeguards can be inserted into an environment, their impacts must be examined in the context of past and anticipated user needs and practices.

So, both to better understand our current requirements and to better employ technological improvements, it is desirable to conduct a closer examination of current practice. The next section provides the first step towards such an examination.

THE WORLD ACCORDING TO USERS

As noted above, two key factors characterize the environment of many DoD computer users:

- Dedicated mode philosophy
- Occasional ineffective use of computer security safeguards

Probably the main reason so many DoD computer systems operate in the dedicated mode is that that is how the manual system operated before the computer was introduced. Much of the DoD has a dedicated mode philosophy. In financial systems, separation of duties and knowledge are usually considered to be the most important security principles, and the compartmentation of knowledge practiced by terrorist cells is well known. The DoD does follow this principle, but the cells sometimes tend to be large. DoD security policy strongly advocates the importance of need-to-know separation. For example, Army Regulation (AR) 380-380 states that "a serious violation potential exists if all users are authorized access to all data"². Sometimes this emphasis is not well reflected in the way mission responsibilities and knowledge are partitioned and assigned, however.

This lack of mission-driven emphasis on separation of duties and knowledge is unfortunate, because computers change the mission equation and can increase the risks involved:

- An office of 20 people might reasonably employ a dedicated mode philosophy for most of its work. A computer network of 500 people cannot.
- It takes awhile and might appear suspicious to reproduce a large classified document on a copying machine. Disks can be copied quickly and without attracting undue attention. Disk contents can be quickly and easily transmitted anywhere in the world using equipment commonly found in homes.

Often dedicated mode is unquestionably the correct operating mode. This is the case for many personal computers and

for systems that truly have no requirements for need-to-know separation. Where dedicated mode is appropriate, it can offer advantages. For example, in a dedicated mode system there is no need to manage security access tables that, if improperly managed, can deny access to authorized users.

On the other hand, with the increasing amount of information being stored in computers and the increasing number of users being granted access through networks, dedicated mode operation is becoming more risky. The management challenge is to recognize when dedicated mode is appropriate and when it is not. The point of this paper is not that dedicated mode operation is inherently desirable or undesirable, but that the decision must be made wisely.

The second factor characterizing the environment of many DoD computer users is occasional ineffective use of computer security safeguards. Perhaps the one thing worse than inadequate security is to have inadequate security and not realize it. Computers can contribute to this misapprehension, because it is easy to forget that computer security is dependent on the people who use and administer the computer. The discussion earlier in this paper notes the existence of cases in which safeguards are not used or are used improperly. This is an aspect of DoD computer misuse that cannot be ignored or assumed away.

Where safeguards are not effectively used, reasons include the following: (1) people make errors and take shortcuts, (2) people have not been adequately trained to use the safeguards, (3) people do not appreciate the importance of computer security safeguards, (4) security resources are insufficient, and (5) technical computer security safeguards can be penetrated. Several words of explanation are warranted to illustrate why insufficient security resources can lead to ineffective use of safeguards.

Whereas industry is free to grow, the DoD is not. In the DoD, it is easier to buy an additional computer than it is to hire an additional person. One argument for buying computers has been that they reduce the number of people needed. Unfortunately, some DoD offices purchase computers only to discover that the opposite is often true. In the security area, policy (e.g., AR 380-380, 1985) states the need for additional security resources by mandating the creation of new roles such as:

- Network Security Officer (NSO)
- Automatic Data Processing System Security Officer (ADPSSO)
- Terminal Area Security Officer (TASO)

People assigned these roles are responsible for such tasks as establishing and maintaining security databases (e.g., user clearances, passwords, and access capabilities; facility security profiles) and maintaining and reviewing system audit information. The problem is that these roles are almost always assigned as additional duties and that the people assigned the roles sometimes have insufficient incentives, time, and training to fulfill them. The impact is that computer security safeguards can become ineffective.

Where the use of internal computer security safeguards is ineffective, the options are either to use the safeguards more effectively or to place less reliance on them. The management challenge is to decide which option is appropriate. In many cases, the latter approach is chosen and the system is operated in dedicated mode. There are many cases, however, when dedicated mode operation will not suffice. Furthermore, even with dedicated mode operation, many information, personnel, physical, communications, and emanations security safeguards are needed. The remainder of this paper presents management actions for improving DoD security in light of the user environment described above.

MANAGEMENT ACTIONS

The management actions for improving DoD computer security are fundamental and can be simply stated: improve both training and field support. Improved training will help DoD personnel to better manage and use systems. Improved field support will enable improved independent checks of field practices, and thereby should also improve system management and use. These actions are not a complete management program - that is beyond the scope of this paper. Nevertheless, the actions are key steps that should be taken.

Training improvements are needed both in computer security training and in overall security training. The need for improved overall security training is fundamental. DoD mission training should provide more emphasis and guidance on separation of duties and knowledge. As more people recognize the importance of need-to-know separation, it will become easier to justify the acquisition and use of need-to-know controls (e.g., discretionary access control mechanisms). Meanwhile, some shared computers offer no more protection than the shared safes of the paper world.

The need for improved computer security training in the DoD is pervasive. It applies equally both to young enlisted personnel (who are often the users, operators, and maintainers) and to senior officers (who are often the planners and

creditors, and sometimes the users as well). Some DoD personnel still do not know that there is more to computer security than TEMPEST. They do not understand the difference between dedicated and system high mode. Some who know a little about security think that the problems will be solved by end-to-end encryption. Others who have heard about the Orange Book know nothing about the advantages of volatile memory or removable hard disks³.

This lack of knowledge could give rise to problems. For example, volatile memory and removable hard disks can provide a periods processing capability (to alternate operation between multiple security levels) and can simplify physical security requirements for the data (since the disks can be locked in safes). If procurements ignore these features, some users might find it difficult to satisfy their security requirements.

The key to a successful computer security training program is to include computer security training as an integral part of both mission and system training. This training will have to overcome the skepticism that some people feel towards computer security requirements, which defend against threats that the people do not consider significant. To overcome this skepticism, training should present convincing examples of why computer security safeguards are needed. These examples should involve easily understood threats such as human error, rather than arcane threats such as Trojan horses or confinement channels.

Personnel turnover in the DoD is high, due to frequent relocations. There is a continuing need to quickly train new users. To accomplish this, computer security fundamentals should be stressed during system familiarization and operation. For example, before being granted initial access to a system, new users should receive a computer security briefing from the ADPSSO. As part of the briefing, users should study and sign a one or two-page statement summarizing the major computer security rules for that specific system, such as the following:

- I understand that the system is authorized to process only data classified Secret or below, and that no Proprietary or Contractor Excluded data may be processed.
- I understand the need to protect my password and agree (1) not to write it down and (2) to change it at least every three months.
- I understand that all output must be treated as Secret, until an approved review procedure determines otherwise.

- I understand that floppy disks may not be removed from the secure area.
- I understand the Red/Black separation requirements for the system. (Simple Red/Black separation guidelines were recently declassified, and should be posted near the system.)

More widespread emphasis on such simple rules would improve computer security practice in the DoD, especially in those situations where users must begin using a system without first having had any formal training. Users cannot be expected to know the prodigious number of rules that constitute DoD computer security policy. Therefore, emphasis should be placed on those few rules that counter the major risks.

The second management action fundamental to improving DoD computer security is improved field support. The day-to-day computer security war is being fought in the field. Yet, with the increasing number of computers being introduced into the DoD, the people in the field are fighting a difficult battle and need reinforcements.

The key people in the field are the computer security managers assigned throughout the DoD. Their role is to oversee the implementation of policy. Unfortunately, the staffing of these offices has not increased commensurate with the increased number of computers being used for classified processing. Some Major Commands with thousands of classified systems have only one person assigned to oversee computer security.

An important part of a computer security manager's job is to coordinate system accreditations. Their review of accreditation packages is often the only independent examination of a system's security. Yet some computer security managers do not have the training or resources to do their job. Since these people could not begin to do the larger job of system certification, typically system buyers, developers, and integrators are relied upon to evaluate their own work.

The result is that every year some DoD computers are placed into operation without adequate security oversight. Some systems are operating with no accreditation at all. The accreditation process is definitely not a meaningless paper process. Computer security managers often find problems during their accreditation review, and system security is usually improved through preparation of an accreditation request. The accreditation process might benefit from some streamlining, but it is an essential process nonetheless.

Several steps can be taken to improve the plight of computer security managers:

- Ensure that all system planners are trained in computer security and that they know to consult with computer security personnel early in the system planning process. If more systems follow the rules, the job of enforcing the rules becomes easier.
- Increase the staffing of field computer security offices. This will be a difficult step, but it is a necessary one.
- Ensure that computer security managers are adequately trained, and give them frequent opportunities to update their training.
- Give computer security managers the rank and recognition their position warrants. Support them in taking punitive action against systems that operate without accreditation or that do not comply with approved approaches.

Some of these improvements in training and field support will be difficult to implement, but efforts must begin. There is a final recommendation that is easier to implement and that should produce near-term improvements: the National Computer Security Center (NCSC) should expand upon its continuing assistance to field support personnel. The NCSC is already providing substantial assistance to the field via such means as travelling training teams. The NCSC could provide further assistance, however, by:

- Conducting a six-month study of field computer security management offices to determine (1) the state of computer security in the field, and (2) what field computer security managers believe is needed (by both themselves in particular and the DoD in general) to improve DoD computer security.
- Sponsoring the development of additional simple management and training tools to improve computer security practice. (The NCSC has already made some useful contributions in this area, such as a one-page summary of personal computer security rules.)
- Encouraging field computer security management people to attend annual NCSC conferences in order to meet each other and to present their views and experiences.

Just as field personnel can benefit from NCSC knowledge, so can NCSC personnel benefit from field experience.

CONCLUSIONS

A brief examination of user environments in the field shows that:

- Dedicated mode operation is the most common mode.
- There is occasional ineffective use of computer security safeguards.

These findings suggest the need for a more thorough study of the state of computer security in the field. Furthermore, the findings must be taken into consideration before new policies or technologies are applied in the field. In some cases the findings represent problems that can readily be solved, but in other cases they might represent fundamental environmental limitations on what is achievable. System managers must be able to distinguish these cases. Technological improvements can be harmful if they result in a false sense of security.

DoD computer security can benefit greatly from improvements in training and field support, which would help us to better manage and use systems. DoD personnel at all levels should be made more informed about computer security, and computer security managers in the field should be given the resources they need to do their job. Now that an improved foundation of computer security policy and technology has been established in the DoD, more attention should be placed on ways to improve practices in the field.

ACKNOWLEDGMENT

The author is grateful for the review and comments provided by LTC L. Steffensen of Headquarters, USAREUR.

REFERENCES

- [1] DoDSI No. 5-86 (September 1986), "ADP Security Deficiencies," Security Awareness Bulletin, DoD Security Institute.
- [2] AR 380-380 (8 March 1985), Automation Security.
- [3] DoD 5200.28-STD (December 1985), Department of Defense Trusted Computer System Evaluation Criteria.

RISK ANALYSIS AND MANAGEMENT
IN PRACTICE FOR THE UK GOVERNMENT

THE CCTA RISK ANALYSIS AND MANAGEMENT METHODOLOGY: CRAMM

Mr Robin H Moses - UK Central Computer
and Telecommunications Agency (CCTA)
Riverwalk House, 157-161 Millbank,
London, SW1P 4RT, England

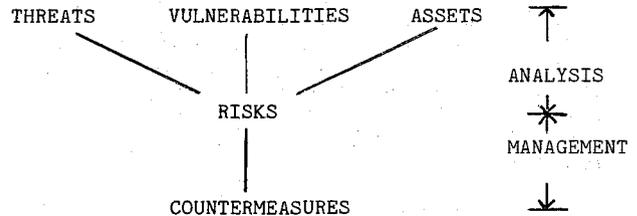
Mr Rodney Clark - BIS Applied Systems Ltd,
20 Upper Ground, London, SE1 9PN, England

INTRODUCTION

1. The paper describes a risk (analysis and) management methodology for Information Technology (IT) Security developed by the UK Government Central Computer and Telecommunications Agency (CCTA) of Her Majesty's Treasury, with the assistance of BIS Applied Systems Limited. The IT Security and Privacy Group of CCTA is the National Authority for advising British Government Departments on all aspects of the protection of IT Systems handling unclassified but sensitive data. The methodology, designed for the identification of justified security measures for both current and future IT systems processing Government sensitive data, has - as of May 1987 - successfully been the subject of five separate trials with systems of different environments. An automated support tool is now being produced, and comprehensive training in use of the methodology by non experts is being prepared.

EXTENT OF THE PROBLEM

2. Her Majesty's Government (HMG) Departments have recognised the general concepts of risk management for some time and implemented them in a pragmatic and relatively subjective manner. However, by mid 1985 both Departments and the Government Security Authorities identified the need to develop a unified approach to risk management which was threat rather than vulnerability driven and which could be applied across the wide range of HMG system types to identify more accurately necessary countermeasures, provide justification for spend and be understandable to non-technically expert general managers. With the rapid expansion of IT and the high cost of development of some secure systems it was not considered to be viable to continue with a significant probability of unjustified spend on security and/or without high confidence that all justified countermeasures had been identified. It was also recognised that the approach would need to cope with the complex situations where many threats could impact more than one asset, many countermeasures could counter more than one threat, and many countermeasures could protect more than one asset. It was agreed that risk management should be put on a much more formal and structured basis to deal with these problems, using as a basepoint the main components of risk analysis and management as shown on the traditional simple model:-



and incorporating related 'sub-components' such as frequency and severity of threats, impacts and countermeasure costs.

HMG APPROACH

3. As a National IT Security Authority for Government Departments, CCTA was invited to mount and manage a project to identify or develop a risk management methodology which would meet thirteen mandatory requirements. These included:-

'able to deal with HMG Operational and Administrative systems of all sizes';

'able to encompass all technical (eg Hardware, Software, Communications) and non-technical (eg Physical, Personnel) aspects of IT security';

'compatible with existing Government IT Security guidance';

'suitable for use during the development of a system, ie for projects as well as existing installations';

'easy to use, after training, by staff with IT but not necessarily IT security experience';

'able to be used such that reviews can be carried out quickly enough to ensure that results are not overtaken by changes in the system';

'able to be used with an automated support tool'.

4. The first task was to examine existing methodologies to determine if any met the HMG requirements. Several methodologies were identified, but none met all the mandatory requirements. Whilst at first glance Annual Loss Expectancy (ALE) based

quantitative approaches seemed attractive, it became evident that the inevitably subjective way in which figures are attributed, particularly costs for data assets, could produce an unsound base and inconsistencies between similar reviews. Also these methodologies typically did not offer much support for countermeasure selection with a consequential need for the reviewer to have IT security knowledge, coupled with the fact that analysis could be lengthy. Existing qualitative methodologies were insufficiently rigorous, did not cover all main components of risk management or were not sufficiently far enough advanced to be of use. Therefore it was decided to devise a new methodology following a qualitative approach, but wherever possible taking quantitative input, and containing no 'hidden' logic.

5. Accordingly, a "manual" version of the methodology has been produced and as of May 1987 has successfully undergone five separate trials encompassing both administrative and operational, and existing and planned, systems. Comprehensive documentation - including management guidelines, the logical design specification for an automated support tool, and an outline of the training course required for its use, have already been produced. Detailed amendments are being incorporated in the documentation, further 'beta' site trials to 'fine tune' the methodology are in progress, and work has started on the production of an automated support tool and a comprehensive training course. CRAMM is now the 'Preferred' methodology for the British Government
Unclassified but Sensitive 'area'.

OVERVIEW OF THE METHODOLOGY

6. The methodology comprises a staged, or modular, approach. The first two stages address analysis of the risk and the third and final one addresses management of risk through the implementation of countermeasures. Each stage is supported by questionnaires and guidelines and sets out to answer one major question. Simply stated these are:-

Stage 1: is there a security need above a certain baseline level?

Stage 2: where and what is the extent of the security need?

Stage 3: how can this need be met?

At the completion of each stage there is a formal management review.

Stage 1

7. The first part of Stage 1 is the important task of precisely determining the nature and boundaries of the system under review, and its various components. This is accomplished by the acquisition of information on the user community and the manner in which they use, or will use, the system - together with an outline system configuration diagram. This information is obtained from interviews with senior installation or project managers, and user managers and their staff, and is essential in providing the reviewer with the understanding necessary for the specific

boundaries of the review to be agreed and later for the questionnaires and guidelines to be put into perspective. It also provides sufficient detail, for instance on the number of 'data owners', for the review to be scheduled. Stage 1 then continues with its major function - the determination of qualitative values for assets, both physical and data. The CRAMM documentation provides detailed advice on how the reviewer should schedule, conduct and record interviews with data owners and personnel responsible for physical assets, and to review results with system or project management. A carefully structured questionnaire enables the reviewer to establish the selection of qualitative values, without 'user bias', for the four possible impacts - disclosure (of data assets), modification (both accidental and deliberate), unavailability (of data assets) and destruction (of physical or data assets). This selection is aided by detailed 'common metric' guidance for data valuation covering such issues as political embarrassment, commercial confidentiality, personal privacy, financial and legal. Physical assets such as hardware and air conditioning plant are first valued on the basis of replacement or reconstruction costs - which are then converted onto the same qualitative scale as that used for data assets. An advantage of the methodology is that time and resource wastage can be avoided where all values are low. In these circumstances what is in effect a shortened version of Stage 2 would be used to check whether there are any threats, vulnerabilities, or combinations thereof, which are of sufficient level to justify greater than baseline protection for low value assets. If the value of all assets is low and only baseline protection is justified, then a review will move directly to Stage 3. Only where asset values are medium or high is Stage 2 recommended. At the end of Stage 1, as with the subsequent two stages, there is a comprehensive management review.

Stage 2

8. The extent of the security needed by a system relates not just to values of assets but also to the levels and nature of threats to which the system could be subjected and the likely vulnerabilities of the system assets to those threats. The first part of Stage 2 is concerned with evaluating the dependency of a system or potential system on certain groups of assets, not all of which are vulnerable to the same potential threats. Then twenty-two generic threat types, for example fire, water damage, system infiltration and misuse of resources, are used as the basis to assess the qualitative threat and vulnerability levels per relevant asset group, using pairs of structured questionnaires incorporating the knowledge of HMG Security experts. As far as possible questions are framed so as to prompt a 'yes' or 'no' answer to avoid 'bias', with each answer afforded a particular score; total scores per questionnaire indicate a high, medium, or low threat or vulnerability. For each relevant asset group, the combination of asset value and assessments of the levels of vulnerabilities and threats are used to calculate a security requirement (ie risk) number on a scale of one (baseline) to five, for each of the four possible impacts, (ie disclosure, modification, unavailability and destruction). At the end of Stage 2,

management has a clear view of the levels of threats to, vulnerabilities of, and thus risks to, particular asset groups. The expression of risks in a numerical form enables direct matching to countermeasures in Stage 3. The completed analysis of risks, ie at end of Stage 2, is reviewed in detail with management before moving to Stage 3.

Stage 3

9. Stage 3 determines how the identified security need can be met, ie countermeasure selection. Taking the determined levels of risk, ie the security requirement numbers, for each asset grouping, countermeasures (covering all aspects of security) are selected from a large 'library' which is referenced by, among other things, security aspect (physical, software, etc) and is further annotated by type, eg reduce risk, reduce impact, detect. If the review is of a current installation, details of existing countermeasures are now recorded. (This activity is deliberately kept until the end of the review to avoid prejudging the effectiveness and/or justification for existing countermeasures). A comparison is conducted to ascertain which additional countermeasures are to be recommended, and which existing ones are not justified. As the list of countermeasures is produced, it is annotated with likely levels of cost (from information held in the 'library'). Then costs specific to the actual or likely equipment types can be added, and a further management review is held.

10. If management is unhappy about some aspect, eg the likely overall cost is outside the budget, "what if" questions can be dealt with (for example, what would be the effect of removing one very sensitive file?). In other words a parameter can be changed and the methodology "re-run". The final step is to determine when a further review should be carried out. Much of the information gathered during the first review can be used in, and thus greatly speed up, subsequent reviews.

PRINCIPAL CRAMM CONCEPTS

Stage 1

11. Stage 1 introduces the first of several concepts used in CRAMM, that of a baseline level of countermeasures which would always be applied to any system. You may think of them if you wish as a 'code of good practice'. For example, for other than truly single user systems the requirement for a user to identify himself to the system during log-on might be defined as a baseline countermeasure. The need for such simple countermeasures is based on the premise that any system must be of some value to the organisation (or why have it?), and therefore needs a certain level of control.

12. Further protection will only be required if the importance of the data to the user or the value of, say, the hardware merits this addition. The principal function of Stage 1 is therefore to establish these values. As mentioned however, Stage 1 initially establishes the scope of the review, and details the system configuration and the manner in which the system is used. Only when a clear picture of the total system has emerged is

the first real risk management task tackled - that of establishing the boundaries of the system under review. Experience has shown this to be an important task and guidelines are given to aid the process. The typical modern system frequently interconnects with other systems which themselves are connected again to further systems. It is important to establish therefore up to which point one is aiming to provide a secure system.

13. The importance of the data can now be assessed by detailed questionnaires directed at the owners and users of data. They are asked to state what the effect on the organisation would be if the data were to be disclosed, modified, made unavailable (loss of service), or destroyed. The reviewer is aided in recording the results of this process by a series of guidelines which enable him to place a value on the data appropriate to the manner in which it is used. For example, if the data contains details of legal contracts, he will ask what the effect would be of the organisation being in breach of contract. Would it be sued? For how much? What would the effect of the publicity be? The guidelines will relate this to a scale of 1 to 10.

14. This approach to establishing the importance of the data to the organisation has been found to have three important advantages:-

- (a) users can much more readily associate with values appropriate to the system; they are not forced to use financial values;
- (b) the relative values that have been established could, if justified, be easily adjusted to an organisation's own perception, without in any way affecting the working of the methodology;
- (c) the use of common guidelines helps to prevent user bias.

15. Asset valuation is completed by listing the replacement or reconstruction costs for hardware, software and environmental facilities. This enables complete understanding of the importance of the system to be obtained and a decision can now be made as to whether it justifies a full scale review, or whether an abbreviated approach could be used. This facility (which is incorporated within the methodology) avoids creating situations in which a great deal of time and money is spent investigating the risks to a system which contains nothing of great value.

16. Stage 1 is completed by a comprehensive review with management to agree the information collected. At this stage discussion usually centres on the extent of the configuration and the user's perception of the importance of the system. These are unemotive topics and consequently agreement can usually be easily reached.

Stage 2

17. The primary function of Stage 2 is to evaluate the level of threats to, and extent of the vulnerabilities of, the system assets. However, another important concept of the methodology is the

recognition that different threats may apply to different parts of the same system. Similarly, vulnerability may not be the same at all points. In practical terms though it would be prohibitively expensive in time and effort and indeed unnecessary to explore the level of threat against every individual asset. Therefore, using CRAMM, assets are grouped in a manner appropriate to the threat. The threat of fire, for example, is likely to vary by physical location and it is therefore appropriate to evaluate this threat against all the assets in one room or small building. However, by comparison, if system infiltration is being considered then the total system could be regarded as the appropriate group of assets since it is normally not practical to separately protect different parts of the same system against this particular threat type.

18. The second part of Stage 2 establishes the security requirement (measure of risk) of each group of assets by relating together the value of the assets (including data), the level of threats to which it is likely to be exposed and the degree of vulnerability. The first of these has been expressed on a scale of 1 to 10 and the other two on a high, medium or low basis. A matrix is used to link the three factors together and express the result on a scale of 1 to 5.

19. The significance of dividing the system into assets or groups of assets becomes more apparent when it is appreciated that the security requirement figure will be used to determine the level of countermeasures. Hence an asset with a high value associated with it may have a higher security requirement than an asset of lower value but the same threat and vulnerability rating. The correct level of protection is therefore established for all parts of the system. Blanket coverage, which frequently results in under or over protection for particular assets, is avoided.

Stage 3

20. Stage 3 is concerned with establishing the countermeasures necessary to meet the security requirement calculated from the analytical work of the first two stages. It therefore moves positively from risk analysis to risk management. This is an area which appears to have received relatively little attention in other methodologies, yet the task of selecting countermeasures is a formidable one. For example, a major installation or network may require several hundred countermeasures to be implemented. These could range from procedures for assigning passwords, to check controls over input data, to encryption, to fire extinguishers in the general office. The range is enormous, making selection extremely difficult.

21. Stage 3 tackles this problem by grouping countermeasures together (countermeasure groups) and relating these to threats. For example, procedural controls over system programmers will relate to the threat of systems infiltration (unauthorised access). The first step, therefore, is to select the appropriate countermeasure groups for each threat. At this stage a considerable degree of overlap is likely to be observed.

Physical access countermeasures, for example, address several threats, (wilful damage, theft, etc). This overlap indicates that these types of countermeasure are likely to be essential.

22. Having selected the countermeasure groups, the reviewer then has access to an extensive list of several hundred countermeasures (arranged under these groups) each of which has been assigned a rating of between 1 (very low, or baseline) and 5 (very high). These ratings correspond to the score calculated when deriving the security requirement, and thus the reviewer can easily select the appropriate countermeasures.

23. For an existing installation, the same list can now be used to examine the previously implemented countermeasures. These are then compared against those identified as necessary by CRAMM and recommendations made where there are discrepancies. While normally the recommendations will address the requirement for additional countermeasures, this is by no means always the case. In some instances in our 'beta' trials, recommendations have been made to consider removing countermeasures which did not seem to be justified.

CONCLUSION

24. Thus to conclude, the main CRAMM concepts are:-

- baseline level of countermeasures;
- 'common metric' guidance for qualitative valuation of data assets for the four major impacts;
- no presumptions made as to the need for previously implemented countermeasures;
- qualitative assessment of threat types against specific groups of assets;
- qualitative assessment of the vulnerabilities of these specific groups of assets;
- combination of qualitative values for assets and threat and vulnerability ratings to form numeric indications of risks;
- matching numeric indications of risks to specific countermeasures;
- for an existing installation identifying not only justified but also unjustified countermeasures.

We feel that these were needed to meet the originally specified criteria for a methodology for the UK Government.

25. Indeed, the 'manual' methodology has been produced and tested and it is evident that, with the use of the automated support tool to considerably reduce review time, it fulfils the specified requirements. Particularly popular with trial site staff has been the 'common metric' guidance for establishing qualitative data values, and the production of lists of specific countermeasures.

26. It is now clear that information collected during a review could be used to identify particular evaluation needs and to construct security policy and requirement documents. Indeed, the methodology will be invaluable to management in presenting easily understandable results in the form of countermeasure lists justified in accordance with the real security need (and for existing installations identifying countermeasures which may not be justified and could be removed - probably with cost saving and easing of operational constraint). Management will thus be able to consider submissions for money spend on security supported by a logical, properly-constructed and justified case.

ROBIN MOSES
CCTA
20 May 1987

It should be noted that the CCTA methodology, CRAMM, is Crown Copyright.

A PANEL DISCUSSION ON
RISK MANAGEMENT: A PLAN FOR THE FUTURE

Dr. Sylvan Pinsky
Senior Scientist

Office of Research and Development
National Computer Security Center
9800 Savage Road
Fort George G. Meade, Maryland 20755-6000
(301)859)4485

ABSTRACT

The federal government and private industry have a long-standing interest in conducting computer security risk analyses. Analysis is part of the larger, more comprehensive "risk management" process which describes the types of approaches and methods that address all activities leading to cost-effective safeguards for automated information systems. Numerous computerized tools have emerged over the last 3 years to assist analysts in completing the risk management process. Each of these models deals with only one aspect of the total process, such as vulnerability assessment, threat assessment, or annual loss expectancy calculation.

There is a significant interest and need in the computer security community to have effective tools, techniques, and guidance for completing the risk management process. The National Computer Security Center and the National Bureau of Standards have jointly sponsored forums for exchanging ideas and presenting approaches to risk analysis. These two organizations have identified the major issues in risk management and have embarked on a plan that describes the steps necessary to resolve the problems, lays the foundation for developing a comprehensive model for risk management, and provides guidelines for conducting the process and selecting effective safeguards for computer systems. The cornerstone of the plan resides with the construction of the conceptual model of the risk management process. This model will describe the interrelationships of the components of risk management (e.g., threats, threat frequencies, vulnerabilities, safeguards, risk, outcomes) in a formal way so that we all have a common understanding of the risk management process. This conceptualization will help explain where alternative methods or approaches fit into the overall process.

The panel activity will begin with a presentation of the elements of the road map for the future of risk management. This discussion will include the conceptual framework, the creation of a risk management laboratory and testbed, case studies, data acquisition, model development, and related topics. Panelists will have an opportunity to critique the plan and present alternative recommendations. The panel will conclude with a 15- to 20-minute question and answer session. Panel membership will consist of Stuart Katzke of NBS, Sylvan Pinsky of NCSC,

Robin Moses and Roger Clark from the United Kingdom, Gene Troy of Martin Marietta, and Kurt Schmucker of Productivity Products, International.

m-EVES

Dan Craigen

Research and Technology
I. P. Sharp Associates Limited
265 Carling Avenue, Suite 600
Ottawa, Ontario K1S 2E1
CANADA

Telephone: (613) 236-9942

(IPSA Conference Paper CP-87-5402-21)

Abstract

This paper reports briefly upon the progress of the m-EVES research and development project. m-EVES is a prototype verification system being developed, under contract, by I.P. Sharp Associates Limited.

1 Introduction

The major goal of the m-EVES research and development project is to design and to implement a program verification system¹ which satisfies the following requirements:

- The system is to be based upon sound mathematics.
- The system is to include state-of-the-art techniques in theorem proving, workstations, compilers, and existing mathematics.
- The system may be used to develop programs required to satisfy NCSC A1+ and UK/Canadian equivalents.

Our project is divided into two distinct phases. In the first phase, we are to develop the m-EVES environment; in the second phase, we are to develop EVES.

m-EVES is to be a research and pedagogical environment that emphasizes program verification concepts. The system will handle a new programming and specification language (called m-Verdi), a new prototype theorem prover (called m-NEVER), sundry workstation ideas, and will have a production quality compiler for m-Verdi.

The essential roles of the m-EVES environment are as follows:

- To be used for instructing our clients about program verification techniques;
- To allow us to test various unproven ideas before committing to a design for the EVES environment; and
- To obtain feedback from the various decisions we have already made. This includes decisions respecting mathematics, language and prover capabilities.

EVES is to be the production quality verification environment. EVES will handle a dialect of m-Verdi (called Verdi), which will have significantly stronger specification and programming structures; and will include a state-of-the-art theorem prover (called NEVER), a collection of specification and program analysis tools, and, of course, various compilers for Verdi.

One immutable requirement of our project is that both m-EVES and EVES must have a sound mathematical basis. We maintain that every verification system should be able to exhibit such a basis; otherwise, one must question the mathematical proofs arising from the system. For example, many of the current (North American) systems do not check whether declared functions are well-defined. An elementary example of an ill-defined function is the following Boolean function:

`Russell(x) is defined as not(Russell(x))`

Such an ill-defined recursion allows one to prove the theorem "FALSE" which then throws into doubt any pretensions of verified software. While such pathological examples are easy to recognize, we have to be concerned about the subtle occurrences of such events. Another paper [Cra 86b] discusses in more detail some of the generic strengths and weaknesses of current verification systems.

Of course, even with such a mathematical basis, there may be unsoundness. As an example of a different kind of unsoundness, consider the incorrect (or incomplete) implementation of the verification system itself. Note, however, that the presence of the mathematical basis opens the door to the possible verification of components of the verification system itself; its absence completely negates such a possibility.

Currently, our project is focussing upon the m-EVES environment. It is expected that the system will be completed by November 1987.

The m-EVES development has generally followed two streams:

- The design of m-Verdi and its underlying mathematics.
- The development of the m-NEVER theorem prover.

In the remainder of this paper, I will discuss briefly each of these streams.

2 m-Verdi Development

The major requirement of the m-Verdi language design is that m-Verdi support the development of verifiable software. By *verifiable* it is meant that rigorous, mathematically sound proofs (that a program is in consonance with its specifications) are possible. To attain the goal of verifiability, the requirements were refined to include the following:

- A formal semantic description of m-Verdi must be presented, and
- A sound logic, for reasoning about m-Verdi programs, must be developed.

¹In this paper, I do not want to spend time discussing the rather lengthy history of the EVES project. Another paper [Cra 86a] discusses the history of the project and the evolution of our thoughts.

The m-Verdi language was designed basically as a “proof of concept” language. We wanted to show that rigorous mathematics could be developed to support the verification process and the languages used as a part of that process. As noted above, when we move onto the second stage of the project, the development of EVES, we will enhance the language with more powerful specification and programming facilities—resulting in Verdi.

The design of m-Verdi has led to a language which is quite different from its Pascal-based forbears, even though many of the same concepts are found; it was a matter of different packaging and appropriate simplification.

I have included, at the end of the paper, an example m-Verdi program. This program has been verified using the m-EVES environment as it existed in early 1987². Since that environment was incomplete (for example, the well-formedness of procedures was not yet implemented), it is possible that an error may have slipped through. I remind the reader about my previous comments relating to unsoundness. (However, this example has also been processed by an m-Verdi compiler and no well-formedness errors were uncovered.)

An m-Verdi compiler has been implemented on a VAX/750 running VMS³. To enhance the retargetability of the compiler, the Code Generator Synthesis System (CGSS) of Karlsruhe University is being used. As a result, we are now one of the (few) sites that is in the position of being able to execute verified code. As a case in point, a simplified version of the Flow Modulator was verified, compiled and then executed on the VAX.

In the following subsection, I have included an edited section of the m-Verdi reference manual [Cra 87] which presents a brief overview of the language.

2.1 m-Verdi Overview

A *declaration* is used to introduce a set of new symbols to a vocabulary and to prescribe properties to these symbols. m-Verdi requires declaration before use and disallows the redeclaration of symbols. There are five different kinds of symbols: Constant symbols, Variable symbols, Type symbols, Function symbols, and Procedure symbols.

A Type symbol denotes a set of values. A Constant symbol denotes a fixed value of a fixed type. A Variable symbol may be used in valuations. A Function symbol denotes a function. A function is a mapping from an n -tuple of values to values of a fixed type. A Procedure symbol denotes a procedure.

An axiom restricts the possible interpretations for the symbols in a vocabulary.

The *Bool*, *Int*, *Char* and *Ordinal* types belong to the initial vocabulary (i.e., the predefined m-Verdi symbols). With each type, a set of literals, and constant, variable and function symbols are defined. The *Bool* type denotes the logical truth values. The *Int* type denotes the set of unbounded mathematical integers. The *Char* type denotes a finite set of graphic symbols. The *Ordinal* type denotes an initial segment of the mathematical ordinals (up to ω^ω). Other types are introduced through an Enumeration type declaration, a Restriction type declaration (which defines a set of values using an explicit *Bool* predicate), an Array type declaration or a Record type declaration.

An Expression is an m-Verdi sentence which may be evaluated (using a vocabulary and valuation⁴) to produce a value

²The system has been significantly modified, since I proved the example, as a result of decisions made during the spring of 1987. We have modified the m-EVES interface so that interaction now occurs through a command language. This point is discussed further in §4.

³VAX and VMS are trademarks of Digital Equipment Corporation.

⁴A valuation is a pairing of variable symbols with values.

of a fixed type. The Expressions are equality, inequality, evaluation of a constant, evaluation of a variable, evaluation of a parenthesized expression, evaluation of a function application, evaluation of a constructor, and evaluation of conditional and quantification expressions.

A Command is an m-Verdi sentence which denotes one or more execution steps and determines, in part, the ordering of the execution steps. It is through the execution of commands that the values associated with a program's observables⁵—and valuations—are modified. The m-Verdi commands are **exit** (from a loop), **return** (from a procedure), **abort** (the program), Assignment command, Annotation, Procedure call command, Conditional command, Loop command, and the Block command.

Certain m-Verdi constructs are used solely for specifying functional relationships. These are the Initial clause, Pre condition, Post condition, Measure condition (used in proofs of termination and well-definedness of recursive functions), Invariant (of a loop) and Annotation (m-Verdi's equivalent of the assert command). [Saa 87] discusses in detail the proof theoretic issues arising from the language.

A Package collects together a sequence of declarations and restricts the availability of certain symbols in the sequence. A Package may be used to support information hiding and abstraction.

An Environment is used to introduce symbols which will form a link between the m-Verdi program and the program's observables. Symbols may also be introduced to support the expression of specifications. The Environment acts as part of the axiomatic basis to an m-Verdi program. The Environment will include the specification of routines which cannot be implemented in m-Verdi but are crucial to its execution and its ability to modify the observables.

2.2 Mathematics and Extensions

The semantic basis of the language is described using a form of Denotational Semantics. There are no real surprises in this part of the work.

Much more interesting problems arose with the development of a logical system for reasoning about m-Verdi programs. In fact, this area required the development a new logical system (by my colleague Mark Saaltink) [Saa 87]. It is worthwhile noting that Predicate Calculus systems are inadequate for reasoning about and specifying programs. For example, the Predicate Calculus does not handle recursive functions nor the introduction of new symbols to the vocabulary of the logic. The logic is based upon Gentzen-style deduction.

Each declaration in an m-Verdi program requires an acceptance proof. For example, recursive functions must be well defined and verification conditions (the acceptance criteria) for procedures, which are generated using a Verification Condition Generator (VCG), must also be proven. The logic has been shown to be sound relative to the Denotational Semantic model and readers should note that, since the VCG is a part of the logic, we have proved that the VCG performs the correct analysis of procedures. The mathematics is completely described in Mark Saaltink's paper [Saa 87].

Some of the intended additions to m-Verdi include polymorphism and higher-order functions. Other additions are rather basic (e.g., for loops and case commands); such facilities were not included in m-Verdi since they were only “quantitatively” interesting, not “qualitatively” interesting. All of these additions will materially improve the expressibility of the specification and programming facilities of the language and will more usefully support the development of reusable mathematical theories.

⁵The visible effect of a program's execution is completely ascertained from the set of observables.

3 m-NEVER

The second major research stream of the project is the development of a new theorem prover. This prover incorporates a number of techniques that are under investigation by the theorem proving community.

The prototype prover, called m-NEVER (Not the EVEs Rewriter), consists of six components: a simplifier (a tautology checker augmented with Nelson-Oppen congruence closure and linear programming techniques); a rewriter (that handles conditional rewriting with backchaining, forward rules, and allows for rules which permute parameters); an invoker (that heuristically expands function definitions); a reducer (that reduces a formula by an innermost-leftmost application of simplification, rewriting and invoking to each of the subexpressions of a formula; the reducer uses a cache to maintain valid reductions and thereby significantly improve its performance); user commands (examples include `split`, `invoke`, `prove`, `undo`, and `try`); and the required support for I/O and database management. An induction mechanism, which is modelled on the approach used by Boyer-Moore, is also included.

The theorem prover supports the interactive development of proofs, but also has powerful automatic tools. For example, there is a command which instructs the prover to bring all of its heuristics to bear (including conditional rewriting and proof by induction) on a proposition. Other commands are much more selective in choosing portions of the prover's capabilities to be applied to a proposition. Users of the prover may instruct the prover whether facts are to be used as lemmata or as forward or backward chaining rewrite rules. The capability for defining rewrite rules can greatly decrease the amount of manual interaction required. The decision to support powerful automatic features and, yet, to allow for selective user control is a fundamental design decision. As the developers of n-NEVER have stated elsewhere [PK 87]:

"Although NEVER provides powerful deductive techniques for the automatic proof of theorems, it also includes simple user steps which permit its use as a system more akin to a proof checker than a theorem prover. . . . It represents a tacit admission that we do not intend to develop a deductive system which is fully automatic; rather, for some proofs, it may be essential for the user to resort to hand steps, since the automatic capabilities may be inadequate.

The result of combining the manual and automatic functions within a single system creates the possibility of a synergy between abilities of the system (fast and accurate) and the user (the necessary insight)."

One of the major goals of this effort is to develop a prover which allows for journal-level inference steps, thereby addressing one of the problems with previous verification systems. A more complete description of the prover may be found in [PK 87].

As an indication of the theorem prover's power, it has been used to successfully prove many of the problems from the Kemmerer Assessment Study of verification systems [Kem 86], Kemmerer's Library Specification [Kem 85], David Gries' T-square [Gri 82] and the consistency of theories describing a sequence theory and a theory of sets.

4 m-EVES Interface

The verification system runs upon Symbolics hardware and, consequently, makes use of the windowing software and graphics packages. It is expected that these facilities will greatly increase the utility of the system. (Either J Strother Moore or Bob Boyer

once told us that the power of the Boyer-Moore prover could be increased by an order of magnitude solely by increasing the bandwidth of information between the prover and the individual using the prover.)

The interaction with m-EVES occurs using a "prover command language" (`pcl`) and may occur using either an EMACS buffer or a Lisp Listener. There are six classes of commands:

- **Goal Commands** – The three commands `retry`, `try` and `try next untried` are used to select a proposition for proof.
- **Proof Steps** – These commands are the basic theorem prover commands for modifying a proposition. Examples have been enumerated previously.
- **Package Commands** – The major unit of abstraction and encapsulation in m-Verdi is the package construct. There are six commands for identifying the beginning and end of a package, the beginning and end of an environment, and the beginning and end of a package model.
- **Database Commands** – These commands elicit information about proof status, information pertaining to various proof events, the undoing of prover events, and the freezing and thawing of the database.
- **Declarations** – These commands are essentially m-Verdi declarations. However, the `pcl` has generalized the m-Verdi axiom declaration to include further information pertinent to the proof process (e.g., trigger expressions for forward rules) and packages are handled differently (as noted above).
- **Miscellany** – These commands are used to reset the prover to its initial state, to begin and end scripting, to quit the prover, and to read in a file of prover commands.

The `pcl` interface is now in place (as of May 1987) but some of the commands are not supported. The commands of particular note are those dealing with the environment and packages. While the environment commands will be easy to handle, some rather significant modifications to the prover must be made to properly handle packages.

The environment will fully support the prover's capabilities for interactive proving. As a result, when one is trying to prove a proposition and notes that some subsidiary facts are necessary, it is possible to introduce the new facts and either prove them immediately, or temporarily assume them. (This facility will no longer be available after changes are made during the summer of 1987. In particular, to simplify the checking for non-circularity of proofs, each declaration must be proven as it is added to the system. The only instance where proofs may be deferred will occur when package headers may be added and the corresponding package body deferred. However, when it is time to add the package body, the prover state will have to be returned to the state occurring after the procedure header was added; subsequently, the package body may be added and the various proof obligations satisfied. The approach of forcing proof when a declaration is being added is similar to the approach used by Boyer and Moore.)

For now, if the program is to be compiled, it has to be transmitted to a VAX (see §2). It is still unclear whether an m-Verdi compiler will ultimately reside on the Symbolics machine and we have yet to consider the issues of incremental compilation.

5 Conclusion

The project will result in four major advances:

- The design and implementation of a programming and specification language which has a complete, formal mathematical basis and supporting logic.
- The design of a sound and complete logical system which is sufficiently powerful to handle constructs used in the verification process.
- A new theorem prover which incorporates many of the state-of-the-art concepts currently under investigation in the theorem proving community.
- The use of workstation technology to enhance the interaction between programmer and verification system.

We have tried to learn from the experiences of the existing verification system efforts (e.g., [Kem 86] [Cra 85] [Cra 86b]). Our development of a solid mathematical foundation allows us to present some strong statements about our efforts and opens the door to verifying components of the verification system. Further, we attempt to decrease the cost of the verification effort, by increasing the power of the theorem prover, environment, and, ultimately, when m-Verdi is strengthened, the specification and programming language.

6 Acknowledgements

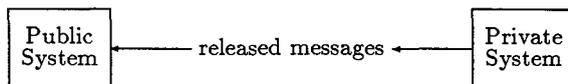
The following individuals have been involved in either the technical or support staff aspects of the project: David Bonyun, Brenda Brown, Sentot Kromodimoeljo, Irwin Meisels, Anders Neilson, Bill Pase, Mark Saaltink, Karen Summerskill and myself, Dan Craigen.

The language was developed by Craigen and Saaltink. The mathematics is due primarily to Saaltink. m-NEVER is being developed by Pase and Kromodimoeljo. Meisels and Neilson are implementing the m-Verdi compiler.

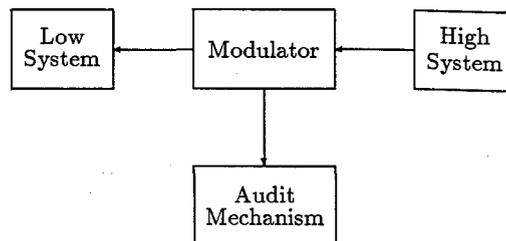
7 Micro Flow Modulator

The rather simple example described here is derived from an Affirm description of a flow modulator which I specified during the Kemmerer Assessment Study [Kem 86][Cra 85].

Suppose we have two computer systems, Public and Private. It is intended that messages will be allowed to flow from the Private system to the Public system if the messages satisfy a particular Boolean predicate defined over messages. (Such a predicate may, for example, check that no sensitive information is being publically disseminated.)



The program described herein, specifies and implements a **Flow Modulator**. The **Flow Modulator** will sequentially read a message from the Private System, determine whether that message satisfies an appropriate Boolean predicate, and based upon the result, will either release the message to the Public System or will log the rejected message. So, the above diagram may be modified slightly to the following:



For the purposes of this exposition, details respecting the format of messages and the definition of the Boolean predicate are ignored. Further, it is specified that the Modulator will process exactly “number_of_messages” messages and then terminate. It is assumed that the I/O channel types are of the same kind.

The following example has been processed by an m-Verdi compiler and has also been verified using an earlier prototype m-EVES verification system. The comments of the form “{! ... }” enclose commands which are recognized by the verification system and are used to prove the proof obligations arising from the associated declarations.

The program is liberally sprinkled with remarks which, hopefully, clarify aspects of the problem being solved and of m-Verdi. Only that text which is printed in typewriter font was presented to the verification system. (Actually, the sequence of declarations was presented—I did not use the **program** and **environment** clauses. The entire program has been processed by the m-Verdi compiler.)

The program is called “micro_flow_modulator.” This name has no effect on the vocabulary.

```
program micro_flow_modulator =
```

The environment is used to introduce names which will form a link between the m-Verdi program and the observables being modified. It also forms the axiomatic basis for the program. There are no (direct) proof obligations involved for the declarations occurring within the environment.

```
environment
```

An unspecified executable type, called “message”, is declared. In this instance, a pragma is used to indicate, to the compiler, that a message will require 1024 bytes and requires a particular word orientation.

```
prog type message =  
    pragma (alignment = 1, size = 1024)
```

The following sequence of declarations introduce a theory of sequences of messages. A complete theory of sequences can be quite rich; what we have here, however, is a basic kernel of sequence theory concepts. An algebraic datatype style of presentation has been used to describe the theory.

The theory of sequences is used to specify (and annotate) our program. Only one declaration, that for e0_message, is required to be an executable declaration.

```
type sequence_message
```

The reader should be aware that the following variable declaration, and all subsequent variable declarations, introduce variable symbols to the vocabulary; a program’s state is not modified by such declarations.

The reader should be aware that the following variable declaration, and all subsequent variable declarations, introduce variable symbols to the vocabulary; a program's state is not modified by such declarations.

```
var i0_message, i1_message, i2_message: int

prog var e0_message: message
var e1_message, e2_message: message

var s0_message, s1_message, s2_message: sequence_message

const empty_message: sequence_message

function tack_message (e0_message, s0_message): sequence_message

function head_message (s0_message): message

function tail_message (s0_message): sequence_message

axiom pragma (rule, name = "head_tack_message")
  all s0_message, e0_message:
    head_message (tack_message (e0_message, s0_message)) = e0_message

axiom pragma (rule, name = "tail_tack_message")
  all s0_message, e0_message:
    tail_message (tack_message (e0_message, s0_message)) = s0_message

axiom pragma (rule, name = "sequence_equality_message")
  all s0_message, s1_message:
    implies (and (s0_message <> empty_message,
                 s1_message <> empty_message),
            (s0_message = s1_message) =
            and (head_message (s0_message) = head_message (s1_message),
                tail_message (s0_message) = tail_message (s1_message)))

axiom pragma (rule, name = "tack_equal_empty_message")
  all s0_message, e0_message:
    not (tack_message (e0_message, s0_message) = empty_message)

function size_message (s0_message): ordinal

axiom pragma (rule, name = "size_tail_message")
  all s0_message:
    implies (s0_message <> empty_message,
            ordinal'lt (size_message (tail_message (s0_message)),
                        size_message (s0_message)))

function length_message (s0_message): int =
  measure size_message (s0_message)
begin
  if s0_message = empty_message
  then 0
  else plus (1, length_message (tail_message (s0_message)))
  end if
end length_message

axiom pragma (name = "length_is_non_negative_message")
  all s0_message: int'ge (length_message (s0_message), 0)
```

```

axiom pragma (rule, name = "length_test_message")
  all s0_message, s1_message:
    implies (length_message (s0_message) <> length_message (s1_message),
             not (s0_message = s1_message))

```

This brings us to the end of the sequence theory kernel. To show that the aforementioned theory is consistent is a task that the specifier of the problem should tackle, not the person who has been presented with the specification and told to implement a program (whose specification is presented in terms of sequence theory). For completeness, I should note that a model for the kernel has been developed (using m-EVES) and, as a result, the kernel theory is consistent.

The unspecified function "ok" will be the function used to check that messages may be released to the Public system. In this case, there are no axioms restricting the possible implementations of "ok". As a consequence, in the extreme cases, "ok" could always return true or always return false and still satisfy the intent of the specification.

```

prog function ok (e0_message): bool

```

"number_of_messages" is to be used as the constant which restricts the number of messages that can be analyzed by the program. The axiom specifies that the value must be positive and bounded by maxint. Consequently, any implementation of the environment must satisfy this requirement.

```

prog const number_of_messages: int

axiom pragma (name = "about_number_of_messages")
  and (int'gt (number_of_messages, 0),
       int'gt (maxint, number_of_messages))

```

The following sequence of declarations, through to the end of the environment, relate to the observables of the program and how they may be modified. In this instance, we have two procedures which are used, respectively, to output a message to some particular port (which will be either a port linked to the Public system or to a port linked to the audit mechanism) or to input a message from the Private system.

With each port we associate a history of the messages that have flowed through the port. An abstraction function, "port_history", is used to capture this intent. From the specifications of the procedures, the reader should be able to conclude that each invocation of the procedures results in the processing of a single message.

```

prog type a_port = pragma (alignment = 1, size = 1024)

prog var port: a_port

function port_history (port): sequence_message

prog procedure output_port (mvar port, lvar e0_message) =
  initial (port'0 = port)
  pre true
  post port_history (port) = tack_message (e0_message, port_history (port'0))

prog procedure input_port (mvar port, pvar e0_message) =
  initial (port'0 = port)
  pre true
  post port_history (port) = tack_message (e0_message, port_history (port'0))

end environment

```

Many of the declarations that follow could just have easily been included in the environment.

The following sequence of declarations are rather specific to the concept of modulator. These declarations make use of the sequence theory abstraction to capture modulator concepts.

“accepted_messages” determines the subsequence of s0_message, preserving order, of elements that satisfy the “ok” predicate. “rejected_messages” is essentially the same function except that it extracts the elements which do not satisfy the “ok” predicate.

Since both these functions are defined recursively, we must show that they do, in fact, describe some function. See [Saa 87] for the proof obligations arising from recursive function definitions. Further, in both instances, a lemma was required. The first step in each proof, introduces the lemma “length_is_non_negative_message”. The second step, prove, results in m-*Never* applying its rewriting and simplification techniques to reduce the formula to true.

```
function accepted_messages (s0_message): sequence_message =
  measure ordinal'val (length_message (s0_message))
  begin if      s0_message = empty_message
    then empty_message
    elseif ok (head_message (s0_message))
    then tack_message (head_message (s0_message),
      accepted_messages (tail_message (s0_message)))
    else accepted_messages (tail_message (s0_message)) end if
  end accepted_messages
  {! use "length_is_non_negative_message" }
  {! prove }
```

```
function rejected_messages (s0_message): sequence_message =
  measure ordinal'val (length_message (s0_message))
  begin if      s0_message = empty_message
    then empty_message
    elseif not (ok (head_message (s0_message)))
    then tack_message (head_message (s0_message),
      rejected_messages (tail_message (s0_message)))
    else rejected_messages (tail_message (s0_message)) end if
  end rejected_messages
  {! use "length_is_non_negative_message" }
  {! prove }
```

The following function is a bool predicate which specifies that every element of a sequence must satisfy the “ok” predicate. The axiom that follows then states that security_property holds over the sequence returned by accepted_messages. This is a rather trivial example of a proof of a specification property. Observe that the proof of “accepted_message_sequence_is_secure” uses automatic induction.

```
function security_property (s0_message): bool =
  measure ordinal'val (length_message (s0_message))
  begin if s0_message = empty_message
    then true
    else and (security_property (tail_message (s0_message)),
      ok (head_message (s0_message))) end if
  end security_property
  {! use "length_is_non_negative_message" }
  {! prove }
```

```
axiom pragma (name = "accepted_message_sequence_is_secure")
  all s0_message: security_property (accepted_messages (s0_message))
  {! prove_by_induction }
```

The following three variable names will be used as formal parameters to the main program and will be directly related to ports over which messages flow to the Public system, to the audit mechanism, and from the Private system, respectively. “number_of_messages_read” will be used within the main program to define a component of the program’s state and will be used as a counter for the number of messages read to some point in time.

```
prog var down, reject, input: a_port
```

```
prog var number_of_messages_read: int
```

The following three functions are used to specify and annotate the main program.

```

function pre_condition (down, reject, input): bool =
  begin and (port_history (down) = empty_message,
            port_history (reject) = empty_message,
            port_history (input) = empty_message)
  end pre_condition

function post_condition (down, reject, input): bool =
  begin and (port_history (down) =
            accepted_messages (port_history (input)),
            port_history (reject) =
            rejected_messages (port_history (input)),
            length_message (port_history (input)) =
            number_of_messages)
  end post_condition

function loop_invariant
  (down, reject, input, number_of_messages_read): bool =
  begin and (port_history (down) =
            accepted_messages (port_history (input)),
            port_history (reject) =
            rejected_messages (port_history (input)),
            length_message (port_history (input)) =
            number_of_messages_read,
            int'ge (number_of_messages, number_of_messages_read),
            int'ge (number_of_messages_read, 0))
  end loop_invariant

```

Finally, the main procedure. The implementation is fairly straightforward. The proof of the procedure required two lemmas, including one referring to `minint` and `maxint`, viz. "MININT-AND-MAXINT-REQUIREMENTS". The "equality_substitute" step results in the replacement of "port_history(input'1)" by an expression it is equated with. As a point of interest, in a later version of the system, when the prover had been augmented with forward rules, the proof of the main procedure was reduced to three steps since the lemmas did not have to be explicitly assumed.

```

main prog procedure flow_modulator (mvar down, mvar reject, mvar input) =
  pre pre_condition (down, reject, input)
  post post_condition (down, reject, input)
  begin
    pvar e0_message
    pvar number_of_messages_read := 0
    loop
      invariant loop_invariant (down,
                                reject,
                                input,
                                number_of_messages_read)
      measure ordinal'val (minus (number_of_messages,
                                number_of_messages_read))
      exit when number_of_messages_read = number_of_messages
      input_port (input, e0_message)
      number_of_messages_read := eplus (number_of_messages_read, 1)
      if ok (e0_message)
        then output_port (down, e0_message)
        else output_port (reject, e0_message)
        end if
    end loop
  end flow_modulator
  {! use "about_number_of_messages" }
  {! use "MININT-AND-MAXINT-REQUIREMENTS"}
  {! prove }
  {! equality_substitute port_history (input'1) }
  {! prove }

end micro_flow_modulator

```

References

- [Cra 85] Dan Craigen. *A Technical Review of Four Verification Systems: Gypsy, Affirm, FDM and Revised Special*. I.P. Sharp Associates Final Report FR-85-5401-01, August 1985.
- [Cra 86a] Dan Craigen. *Program Verification at I.P. Sharp Associates*. I.P. Sharp Associates Technical Report TR-86-5420-04, September 1986.
- [Cra 86b] Dan Craigen. *Some Comments on Program Verification Systems*. To appear in the proceedings of the "Symposium on Safety and Security", (October 20 - October 24, 1986), Glasgow, Scotland. Proceedings to be published by Blackwells. Also I.P. Sharp Associates Technical Report TR-86-5420-05, December 1986.
- [Cra 87] Dan Craigen. *A Description of m-Verdi*. I.P. Sharp Associates Technical Report TR-87-5420-02, June 1987.
- [Gri 82] David Gries. *A Note on the Standard Strategy for Developing Loop Invariants and Loops*. Cornell University Technical Report TR-82-531, October 1982.
- [Kem 85] R. Kemmerer. *Testing Formal Specifications to Detect Design Errors*. IEEE Transactions on Software Engineering 21(1), January 1985.
- [Kem 86] R. Kemmerer, et al. *Verification Assessment Study Final Report, Volumes I-V*. National Computer Security Center C3-CR01-86, March 1986.
- [PK 87] Bill Pase, Sentot Kromodimoeljo. *NEVER: An Interactive Theorem Prover*. I.P. Sharp Associates Conference Paper CP-87-5402-20, January 1987.
- [Saa 87] Mark Saaltink. *The Mathematics of m-Verdi*. I.P. Sharp Associates Technical Report FR-87-5420-03, June 1987.

The Bell-LaPadula Computer Security Model Represented as a Special Case of the Harrison-Ruzzo-Ullman Model

Paul A. Pittelli

ABSTRACT

Currently most computer security models are classified among the three types; access control, information flow, and non-interference. Within the realm of access control lies the classical Bell-LaPadula model. A BLP model consists of a set of subjects and objects, three security level functions, and a discretionary access matrix together with a set of rules used to manipulate the current state of the model. Security in this model is dependent upon the satisfaction of the three properties: simple security, discretionary access, and the * property. An HRU model consists of an access matrix and a finite set of commands which act as matrix transformations. Here security is determined by looking for the existence of an access right in a specific cell of the matrix. We define a specific HRU model (called the Bobo model) and establish a correspondence between the Bobo commands and BLP rules, also between the Bobo and BLP states. Furthermore we observe that this correspondence is security preserving in the fact that a BLP access triple is secure if and only if that access is contained in a specific cell of the Bobo access matrix.

Introduction

The purpose of this note is to show that the Bell-LaPadula model for access control is simply a special case of the not so well known Harrison-Ruzzo-Ullman model. The HRU model consists of an access matrix together with a finite set of commands that are used to manipulate the matrix. In order to develop a model equivalent to BLP's, we need to exhibit commands that are "identical" to the BLP rules.

Before we begin defining the commands, we must first exhibit a correspondence between the "subjects" and "objects" in the BLP model and the "subjects" and "objects" in the HRU environment. The reason for the above quotes is that subjects and objects are disjoint sets in BLP, whereas the set of subjects is contained in the set of objects under HRU. This distinction, though seemingly small, is well worth remembering. However, a key factor of the BLP model is the use of the functions $f_S, f_C,$ and f_O . These functions associate a security value to a subject or object, which allows one to compare subjects to objects.

The preceding paragraph indicates some of the differences we need to consider when relating BLP to HRU. The major concept in this new model is the notion of a subject(object) represented by a set of entities. Defining a subject as a set of sub-subjects allows us to implement the multilevel capabilities of a BLP subject in an access matrix. Likewise an object viewed as a set permits the use of an upgrade command.

Specifically, suppose we have a BLP model which consists of the following entities;

$\Sigma = \{s_i : i=1,2,\dots,k\}$ -- set of subjects

$\hat{O} = \{o_j : j=1,2,\dots,n\}$ -- set of objects; recall $\Sigma \cap \hat{O} = \emptyset$

$L = \{l_t : t=0,1,\dots,T\}$ --set of security values forming a lattice under the partial ordering \geq referred to as the dominance relation. Without loss of generality let l_0 and l_T denote the least upper bound of L and the greatest lower bound of L respectively.

$f_S : \Sigma \Rightarrow L$ -- function yielding the maximum security level for a subject.

$f_C : \Sigma \Rightarrow L$ -- function yielding the current security level for a subject.

$f_O : \hat{O} \Rightarrow L$ -- function yielding the security value of an object.

$R = \{\text{append,write,read,execute}\}$ -- set of access rights.

$M = k \times n$ matrix with $m_{ij} \subseteq R$ representing the set of discretionary access rights that subject s_i has to object o_j .

We now begin showing how to incorporate the BLP model into an HRU environment. First we define the following components of an HRU model:

$S = \{s_i | t : s_i \in \Sigma \text{ and } t \in ST_i\} \cup \{s_0 | T\}$. Corresponding to each BLP subject s_i is a subset S_i of S , where $S_i = \{s_i | t : t \in ST_i\}$, which represents s_i together with all of s_i 's allowable security values. That is $\{l_t : t \in ST_i\} = \{l_t \in L : f_S(s_i) \geq l_t\}$. Formally the elements of S_i come from the cross product space $\Sigma \times \hat{O}$, but for ease of notation we will write the elements as $s_i | t$. Thus $s_i | t$ will denote an HRU subject. Furthermore we reserve the subject $s_0 | T$ to be a system subject. Thus $S_0 = \{s_0 | T\}$ and $ST_0 = \{T\}$. The purpose of $s_0 | T$ is to let the system know at what level an object is currently classified as will be formalized later.

$O = S \cup \hat{O}$ where $\hat{O} = \{o_j | u : o_j \in \hat{O} \text{ and } u \in OT_j\}$. We relate to each object o_j a set of values $\{l_u : u \in OT_j\}$ which represents all the security values that o_j could assume. That is $\{l_u : u \in OT_j\} = \{l_u \in L : l_u \geq f_O(o_j)\}$.

$A = \{\text{active,own,r,a,w,e}\}$ set of generic access rights.

$P = (P[s,o])$, $p \times (p+q)$ matrix where $P[s,o] \subseteq A$ for $s \in S$ and $o \in O$. Here

$$p = \sum_{i=0}^k |ST_i| \quad \text{and} \quad q = \sum_{j=1}^n |OT_j|$$

P is simply referred to as the access matrix.

Primitive Operations:

In the BLP model the rules allow for the creation and deletion of objects as well as the insertion and removal of an access right from a cell in the access matrix M. In the HRU model there are counterparts of these actions which are termed primitive operations. Because of our particular example there is an additional operation: delete a proper subset of the set $O_j = \{o_{jl_u} : u \in OT_j\}$. This last primitive operation will provide the means to implement an upgrade command.

Given system state (S,O,P) we define a primitive operation op as a function $op:(S,O,P) \Rightarrow (S^*,O^*,P^*)$ where:

(1) $op = \text{create object } o_{jl_u}$,

where $O_j \neq O$. We have for all $l \geq l_u$:

$S^* = S, O^* = O \cup \{o_{jl_u}\}$,

$P^*[s,o] = P[s,o]$ for all $(s,o) \in S \times O$

$P^*[s,o_{jl}] = \emptyset$ for all $s \in S$.

(2) $op = \text{delete object } O_j$,

where $O_j \subseteq O \setminus S$. We have for all $l \in L$:

$S^* = S, O^* = O \setminus \{o_{jl}\}$,

$P^*[s,o] = P[s,o]$ for all $(s,o) \in S \times O^*$.

(3) $op = \text{delete objects } o_{jl_u}$,

where $o_{jl_u} \in O \setminus S$. We have for all $l \not\geq l_u$,

$S^* = S, O^* = O \setminus \{o_{jl}\}$,

$P^*[s,o] = P[s,o]$ for all $(s,o) \in S \times O^*$.

(4) $op = \text{enter } x \text{ into } P[S_i, o_{jl_u}]$,

where $x \in A, S_i \subseteq S$, and $o_{jl_u} \in O \setminus S$. We have for all l, l_t with

$$l_u \geq l \text{ and } \begin{cases} l_t \geq l & \text{if } x = r \\ l_t = l & \text{if } x = w \\ l \geq l_t & \text{if } x = a \\ l_t = l_T & \text{if } x = \text{active} \\ \emptyset & \text{if } x = e \text{ or } \text{own} \end{cases}$$

$S^* = S, O^* = O$,

$P^*[s,o] = P[s,o]$ for all $(s,o) \neq (s_i, l_t, o_{jl})$

$P^*[s_i, l_t, o_{jl}] = P[s_i, l_t, o_{jl}] \cup \{x\}$.

(5) $op = \text{delete } x \text{ from } P[S_i, o_{jl_u}]$,

where $x \in A$. We have for all l, l_t with $l_u \geq l$,

$S^* = S, O^* = O$,

$P^*[s,o] = P[s,o]$ for all $(s,o) \neq (s_i, l_t, o_{jl})$

$P^*[s_i, l_t, o_{jl}] = P[s_i, l_t, o_{jl}] \setminus \{x\}$.

As an aid to understanding the effects of the primitive operations on the matrix P, it is helpful to consider that there corresponds to each subject, object pair (s_i, o_j) a submatrix P_{ij} whose rows are indexed by ST_i and whose columns are indexed by OT_j . The consequences of applying the primitive operations can be summarized as follows:

(1) $op = \text{create object } o_{jl_u}$

This operation creates a set of matrices $\{P_{ij} : 0 \leq i \leq n\}$, where P_{ij} has rows corresponding to elements in ST_i and

columns corresponding to the members of OT_j . The cells of each submatrix are all empty.

(2) $op = \text{delete object } O_j$

This operation removes from the matrix P all those submatrices P_{ij} for $0 \leq i \leq k$. Recall that k is the cardinality of S, the set of BLP subjects.

(3) $op = \text{delete objects } o_{jl_u}$

This operation removes a subset of columns from each matrix P_{ij} , $0 \leq i \leq k$, specifically all those columns corresponding to o_{jl} where $l \not\geq l_u$.

(4) $op = \text{enter } x \text{ into } P[S_i, o_{jl_u}]$

This operation inserts x into a subset of positions of the matrix P_{ij} defined by the various values of x.

(5) $op = \text{delete } x \text{ from } P[S_i, o_{jl_u}]$

This operation deletes x from all entries in those columns of P_{ij} corresponding to l where $l_u \geq l$.

Commands:

An HRU command is simply a conditional IF (expression 1) THEN (expression 2) where expression 1 is a boolean function and expression 2 is a sequence of primitive operations. To implement the BLP model in an HRU environment we will use the following commands. For ease of notation let $S_r =$ requesting subject and x a member of the set $\{r, a, w, e\}$.

(1) Command GIVE(S_r, S_i, x, o_{jl_u})

IF own $\in P[s_r, l_t, o_{jl_u}]$ for any l_t , and
active $\in P[s_0, l_T, o_{jl_u}]$

THEN enter x into $P[S_i, o_{jl_u}]$.

(2) Command RESCIND(S_r, S_i, x, o_{jl_u})

IF own $\in P[s_r, l_t, o_{jl_u}]$ for any l_t

THEN delete x from $P[S_i, o_{jl_u}]$.

(3) Command GENERATE(S_r, o_{jl_u})

IF TRUE

THEN create object o_{jl_u} ,
enter active into $P[S_0, o_{jl_u}]$,
enter own into $P[S_r, o_{jl_u}]$.

(4) Command DESTROY(S_r, o_{jl_u})

IF own $\in P[s_r, l_t, o_{jl_u}]$ for some l_t

THEN delete object O_j .

(5) Command UPGRADE($S_r, o_{jl_u}, l_{u_0}, l_{u_1}$)

IF own $\in P[s_r, l_t, o_{jl_{u_0}}]$ for some l_t , and
active $\in P[s_0, l_T, o_{jl_{u_0}}]$

THEN delete objects $o_{jl_{u_1}}$,
enter active into $P[S_0, o_{jl_{u_1}}]$.

Note: For the rest of this paper the sets S, O, access matrix P, and the five commands defined above will comprise that which will be called the Bobo model.

Equivalence to BLP:

The method that we will use to exhibit an equivalence between the BLP and Bobo models is a two-fold process. First we will prove a theorem that will show every state of a BLP

model is achievable by the Bobo model. Secondly a correspondence between the state transitions of the two models will be drawn by simply listing each BLP state transition together with its counterpart Bobo state transition.

Besides showing that every BLP state is achievable by the Bobo model, the theorem below illustrates how to identify the BLP set of secure access triples in the HRU environment. One of the hypotheses of the theorem is an assumption on the initial access matrix P^0 . We assume that for $0 \leq i \leq k$; $1 \leq j \leq n$; $t \in ST_i$; $u \in OT_j$;

$$P^0[s_i l_t o_j l_u] = \begin{cases} \{active\} & \text{if } s_i = s_0 \text{ and } l_u = f_O(o_j) \\ \{own\} & \text{if } s_i \text{ created } o_j \\ \emptyset & \text{otherwise} \end{cases}$$

In order to see that this is not an unreasonable assumption, consider what happens when we generate an object o_j . Suppose in the BLP model subject s_i created object o_j at level $f_O(o_j) = l_u$. In the Bobo model this is accomplished by issuing the command $GENERATE(S_i, o_j, l_u)$. Upon execution of the command we see that the submatrices P_{aj} are all empty except when $a = i$ in which the matrix P_{ij} contains $\{own\}$ in every cell. Also there is only one entry in P_{0j} which is nonempty and that is $P^0[s_0 l_t o_j f_O(o_j)] = \{active\}$. Thus we see that if the system knows who created each object then we can generate the initial matrix P^0 by a sequence of $GENERATE$ commands. Hence our assumption on the matrix P^0 can be made without loss of generality.

Theorem: Let M, f be a state of a BLP model with subjects $\{s_1, s_2, \dots, s_m\}$ and objects $\{o_1, o_2, \dots, o_n\}$. Let β be the set of all possible secure triples (s, o, x) completely determined by M and f . Define a Bobo access matrix P^0 to be, for $0 \leq i \leq k$; $1 \leq j \leq n$; $t \in ST_i$; $u \in OT_j$;

$$P^0[s_i l_t o_j l_u] = \begin{cases} \{active\} & \text{if } s_i = s_0 \text{ and } l_u = f_O(o_j) \\ \{own\} & \text{if } s_i \text{ created } o_j \\ \emptyset & \text{otherwise} \end{cases}$$

Then there exists a sequence of commands c_1, c_2, \dots, c_k such that $(S^0, O^0, P^0) \Rightarrow (S^1, O^1, P^1) \Rightarrow \dots \Rightarrow (S^k, O^k, P^k)$ and $(s, o, x) \in \beta \Leftrightarrow x \in P^k[sf_C(s), of_O(o)]$.

Pf: For every object o_j we perform the following: Suppose s_d is the creator of o_j , then for all $x \in m_{ij}$ issue the command $GIVE(S_d, S_i, x, o_j, f_O(o_j))$, for $i = 1, 2, \dots, m$. Since the set of subjects, objects, and access rights are finite sets then we have generated a finite sequence of commands say c_1, c_2, \dots, c_k which transforms $(S^0, O^0, P^0) \Rightarrow (S^k, O^k, P^k)$.

Claim: $(s, o, x) \in \beta \Leftrightarrow x \in P^k[sf_C(s), of_O(o)]$.

Pf: Suppose $(s_i, o_j, x) \in \beta \Leftrightarrow$

$$x \in m_{ij} \text{ and } \begin{cases} f_S(s_i) \geq f_O(o_j) \text{ and } f_C(s_i) \geq f_O(o_j) & \text{if } x = r \\ f_S(s_i) \geq f_O(o_j) \text{ and } f_C(s_i) = f_O(o_j) & \text{if } x = w \\ f_O(o_j) \geq f_C(s_i) & \text{if } x = a \end{cases} \Leftrightarrow$$

$GIVE(S_d, S_i, x, o_j, f_O(o_j))$ is executed, where s_d is the creator of $o_j \Leftrightarrow x \in P^k[sf_C(s_i), of_O(o_j)]$. •

Now we need to show that all the possible state transitions in the BLP model are attainable in this new setting. Referring to [1] we find that there are 11 state transitions(rules). For each rule we will establish an equivalent command and/or a reason why the rule is satisfied.

Rules 1-5: get, release read/append/write/execute

It is unnecessary to implement a get or release command in our Bobo model. In BLP, a subject has to request get access to an object so that the * property is never violated. However in the Bobo model the enter access primitive operation assures that the * property will not be violated. Thus a subject in the Bobo model obtains access to an object whenever the owner of the object has given him the desired access by executing a GIVE command.

Rule 6: give - read/append/write/execute

This rule corresponds to the command GIVE. The rule checks to see if the requesting (giving) subject has the authority to "give" another subject access to a specific object. This is accomplished by the command via the condition that the requesting subject have "ownership" of the object in question. Furthermore upon a true condition, the given access right is granted so that the BLP * property is not violated. (e.g. If s_i is granted w access to o_j , then w is only added to the set in positions $(s_i l_t, o_j l_u)$ for all $t \in OT_j$.)

Rule 7: rescind - read/append/write/execute

The command RESCIND performs the inverse of GIVE as does the rule rescind in the BLP sense. Again the condition tests the authority of the requesting subject via "ownership" and upon valid authority removes the specified access from the subject's access columns.

Rule 8,9 create, delete object

Commands GENERATE and DESTROY correspond to the rules create and delete respectively. GENERATE creates an object and defines the initial "owner" of the object to be its creator. DESTROY checks for "ownership" for authority to delete the object from the system.

Rule 10: change-subject-current-security-level

There is no need for a command which changes a subject's current security level. The reason is that the Bobo model defines a subject S_i to be the set $\{s_i l_t; t \in ST_i\}$. This allows a subject to work on an object $o_j l_u$ in mode x if and only if there is some $t \in ST_i$ such that $x \in P[s_i l_t, o_j l_u]$. Thus the command automatically changes a subjects current security level to accommodate the desired access to an object.

Rule 11: change-object-security-level

The command UPGRADE performs the function of changing the security level of an object. In BLP the reference monitor needs to verify that the new level is a valid one and that the requesting subject has the authority to change the function f_O . This is accomplished by the "own" access right and the delete objects primitive operation. Furthermore suppose that object o_j gets upgraded from level l_u to l_v . Notice that because of the primitive operation enter, if subject s_i had access x to $o_j l_u$ then s_i will still have access x to $o_j l_v$ provided that the * property is not violated. This implies that the UPGRADE command automatically cancels all current accesses that violate the * property at an objects new security level.

Remarks:

Even though the Bobo model can simulate the BLP model, there are several characteristics that have been created to accomplish this. The first and most important is the new idea of an access right called "own". The BLP model contains a tree structure for the objects called a hierarchy. However the hierarchy of objects is not related to the security of the model

but exists merely because of the application of BLP to the Multics system. Thus we see that the only concept of ownership of an object in BLP lies in the Give function for rule 6. In order to implement this Give function it is necessary to use an "own" access right. The Bobo model is conservative in the fact that the only subject who can have "own" access to an object is that object's creator. However if there is need for group ownership of an object the conditions of the GIVE command can be changed to accommodate this feature.

The other new access right in the Bobo model is "active". The purpose of "active" is simply to let the reference monitor know the current level of an object. This feature then allows for the upgrading of an objects security level.

The use of sets for representing subjects and objects creates more responsibilities for the reference monitor in the Bobo model. In particular, since a subject can work at any level he dominates and an object can assume various security values, then it should be the job of the reference monitor to inform the subject at what level he is working and the value of the object that he is accessing.

The last remark that we want to make concerns the form of the access matrix P. The easiest observation to make is that the submatrix whose rows and columns are indexed by the subjects is completely empty. This reflects the fact that subjects are not allowed to access other subjects in the BLP model. Also the method of giving subjects access rights creates alot of redundant information. That is, if the reference monitor wants to check to see if subject s_i has access x to object o_j then the only cell necessary to examine is $P[s_i]_u, o_j]_u$.

Safety:

This section discusses the concept of safety as introduced by Harrison, Ruzzo and Ullman. Intuitively a "safe" security model (i.e. protection system in HRU terminology) is one which will not allow unauthorized access to objects. The following two definitions formally state the idea of safety.

Def: Given a protection system, we say a command α leaks generic right r from configuration $Q = (S, O, P)$ if α , when run on Q , can execute a primitive operation which enters r into a cell of the access matrix which did not previously contain r .

Def: Given a particular protection system and generic right r , we say that the initial configuration Q_0 is unsafe for r (or leaks r) if there is a configuration Q and a command α such that

- (1) $Q_0 \Rightarrow Q$ by a sequence of primitive operations,
- (2) α leaks r from Q .

The first observation one can make is that any system which utilizes the primitive operation enter will most likely be deemed unsafe. Harrison et al. make the convention that to check for unauthorized leakage, we need to eliminate from testing those subjects who are actually authorized to give (leak) rights. They use the term "reliable" subjects to mean the set of subjects who are authorized to grant the generic right r of an object to another subject.

The general question of whether or not an arbitrary protection system is safe was shown to be undecidable by HRU. However if a specific model consists of commands which involve only one primitive operation (mono-operational in HRU terminology) then the question of safety is decidable. Our Bobo model happens to live in the middle ground of the previous

sentences. That is, the Bobo model is not a mono-operation system but we will show that the model is "safe".

Its not too hard to see that every command except the GENERATE command contains a check for ownership in the condition. Since a subject who owns an object is deemed reliable then the only command in question is the GENERATE command. However, anyone who creates an object is by nature reliable with respect to that object. So we can view the entering of the own access right by the creator of the object giving himself access. Therefore, all the commands in the Bobo model preserve safety which implies the Bobo model is itself "safe".

Conclusions:

Besides an attempt at unifying some of the existing access control models, the Bobo model reveals an interesting point. This is we see that the HRU model is a very general access control model. Moreover one can appreciate the elegance of the model by the fact that the complex BLP model can be defined by an access matrix together with a set of five commands. Being able to incorporate the three functions f_s, f_c, f_o , the BLP discretionary access matrix M , and the set of all current accesses b into the matrix P , allows one to see the interplay between the different security levels and the * property. Also note that we have only dealt with the rules defined by volume 4 of the Bell-LaPadula model. A current topic of discussion is whether or not the rules constitute a part of a BLP model. This is no concern of this paper so we do not attempt to imply either case. However what we show is how the rules of any BLP model correspond to commands in an HRU model. Thus if any more rules are added to the existing BLP model then a new command can be added to the Bobo model accordingly in order to preserve the equivalence. Even though implementation of the BLP model might be simplified by using the Bobo model, the intent of this paper and follow-ons is to try to unite all the existing access control models in one framework; maybe that of the Harrison, Ruzzo, Ullman Model.

References:

- (1) Bell, D.E. and LaPadula, L. J. Secure Computer Systems, Vol. IV: Unified Exposition and Multics Interpretation. MITRE Corp. Tech. Rep. MTR 2997, 1975.
- (2) Harrison, M. A., Ruzzo, W.L. and Ullman, J. D. Protection in Operating Systems. Communications of the ACM, Vol. 19, No. 8, August 1975

COMPARING SPECIFICATION PARADIGMS FOR SECURE SYSTEMS: GYPSY AND THE BOYER-MOORE LOGIC

Matt Kaufmann*
William D. Young

The Institute For Computing Science and Computer Applications
The University of Texas at Austin
Austin, Texas 78712

The Gypsy Verification Environment (GVE)^{1,2} is one of two systems endorsed by the National Computer Security Center for use in meeting the verification requirements for an A1 level evaluation as outlined in the *Trusted Computer Systems Evaluation Criteria*³. Gypsy has been used extensively in secure systems specification and verification projects including the Encrypted Packet Interface⁴, Message Flow Modulator⁵, Honeywell SCOMP⁶, Honeywell SAT⁷, and ACCAT Guard⁸. The Boyer-Moore theorem prover has also seen extensive use in the security arena. It has been used as a component of the HDM verification system⁹ on KSOS¹⁰, SCOMP, and SADCIN¹¹.

Yet the ways in which these two systems are currently used in secure system development efforts are quite different. The GVE is utilized as a fully integrated verification environment. The Gypsy language is used for constructing code and specification; verification conditions are generated and proved in the GVE proof checker; and, in some cases, the Gypsy code is compiled and run. The Boyer-Moore system, on the other hand, is used only as the proof checker for verification conditions generated from specifications written in some high level language such as Special. This is true despite the fact that the Boyer-Moore logic contains a fully executable functional programming language. The Boyer-Moore system has been used not only to state and prove theorems in traditional mathematical domains such as number theory and recursive function theory, but also to specify and prove the correctness of a microprocessor¹², and is being used to prove the correctness of an operating system and a compiler¹³. A main contention of this paper is that the Boyer-Moore logic can also be used effectively as a specification language for secure systems, particularly at the model level.

This paper investigates the viability of the Boyer-Moore logic as a specification language for secure system modeling efforts by comparing it to Gypsy on a significant example. The example we chose was the Low Water Mark problem, a simple secure system which has been used in two different studies^{14,15} for comparing verification systems. At least three different Gypsy specifications for this problem have been published^{14,16,17}; our specification differs from each of these. Using a non-interference style characterization of security, we specified the Low Water Mark system in Gypsy and in the Boyer-Moore logic. The key security theorem was proved in each system using the associated proof-checker. We compare the specifications and proofs in the two languages, point out the advantages and disadvantages of each system, and investigate the possibility of defining a hybrid language which combines most of the advantages of each.

THE TWO LANGUAGES

Gypsy

Gypsy is a descendent of Pascal. It is a unified programming and specification language with facilities for exception handling, data abstraction, and concurrency. The specification component of the language contains the full expressive power of the predicate calculus. Specifications may be written as Floyd-Hoare style program annotations, algebraic-style axioms, or state machine descriptions.

Gypsy is a procedural language but contains a sizable functional component. The specification described in this paper is written entirely within the functional subset of the language. This is typical for abstract specifications. Implementations are usually given in a procedural style.

The following is a fully specified Gypsy implementation of a factorial routine. The *entry* and *exit* assertions in the definition of *F* give its specification. Notice that the function *fact* is a specification function against which the code is verified.

```
scope factorial_example =
begin

function F (n: integer): integer =
begin
  entry N ge 0;
  exit result = fact (N);
  var i: integer := 1;
  result := 1;
  loop
    result := result * i;
    if i = n
      then leave
      else i := i + 1
    end {if}
  end {loop}
end; {function F}

function fact (n: integer): integer =
begin
  exit result =
    if n le 0
      then 1
      else n * fact (n-1)
    fi;
end; {function fact}

end; {scope}
```

Gypsy is fully described in¹ and a methodology for using the language effectively is documented in².

The Boyer-Moore Logic

The Boyer-Moore logic is a quantifier-free constructive first-order logic with equality and rules for defining recursive functions. The language is a minor variant of Pure Lisp¹⁸ and consists of variables and function names combined in a prefix notation. Predicates are represented as boolean-valued functions. Though untyped, the logic supports a restricted version of user-defined recursive data types (the "shell principle"). Despite the absence of quantifiers in the logic, the system allows one to prove lemmas that are, in effect, treated as universally quantified statements.

A Boyer-Moore specification of the factorial function has the form

```
Definition
(ZEROP N)
=
(OR (EQUAL N 0)
(NOT (NUMBERP N)))
```

```
Definition
(FACTORIAL N)
=
(IF (ZEROP N)
1
(TIMES N (FACTORIAL (SUB1 N))))
```

*Supported by ONR Contract N00014-81-K-0634 and Department of the Navy Contract N00039-85-K-0085

**The other is FDM, which will not be discussed further in this paper.

The Boyer-Moore *definition principle* guarantees that functions accepted by the Boyer-Moore system are total. Thus, (FACTORIAL N) is defined even if N is not a numeric argument. This contributes to a specification style which is quite different than that used in Gypsy. The logic is fully described in^{19, 20}.

The Main Differences

The primary differences in the two languages are summarized below.

1. Syntactically the two languages are quite different. Gypsy syntax is Pascal-like mixed infix/prefix notation; the Boyer-Moore logic uses a LISP-like prefix syntax.
2. Gypsy provides procedures, including concurrent procedures. The Boyer-Moore logic is purely functional.
3. Because of the procedural aspects of the language, the semantics of Gypsy²¹ is significantly more complex than that of the Boyer-Moore logic. The Boyer-Moore logic has a simple applicative semantics; in a certain sense, an interpreter for the logic can be defined fairly easily within the logic.
4. Gypsy encourages a top-down development style by allowing the programmer to leave implementations pending. This permits references to and proofs about routines which are not fully elaborated. In the proof domain, there is no enforced constraint that lemmas be proved before they are used. The Boyer-Moore logic encourages a bottom-up development style since functions must be accepted before they can be referenced. Top-down development of proofs can be accomplished in the Boyer-Moore framework by adding lemmas as axioms and then later redoing the proof. However, this is counter to the paradigm of proof development in the Boyer-Moore system; it is assumed that lemmas will be proved before they are used.
5. Gypsy data typing restricts syntactically the passing of arguments to routines. However, there is at present no guarantee that routines will be defined even for arguments of permissible type. The Gypsy Verification Environment supports *partial correctness* proofs; that is, proofs of termination must be performed outside the system. The Boyer-Moore definition principle guarantees that all functions are total. Arguments which are not of the expected *type* are usually treated as if they were. Thus the Boyer-Moore function FACTORIAL above treats a non-numeric argument as if it were zero.
6. The Gypsy specification language contains the universal and existential quantifiers. The Boyer-Moore logic is constructive and quantifier free. Lemmas involving variables are regarded as implicitly universally quantified. To obtain the effect of an existential quantifier it is necessary to provide a "witness function" which computes the required value.

THE LOW WATER MARK PROBLEM

The Low Water Mark problem was introduced in¹⁴ and used there for a comparison of four verification systems. It was recently revived as one benchmark problem for a more extensive comparison of verification systems reported in¹⁵. In the context of that study, two distinct solutions to the problem were coded in Gypsy^{16, 17}. We describe a third solution, coded in Gypsy and in the Boyer-Moore logic and fully verified in each of the two systems. The method of specification follows the *non-interference* approach described in the following section.

Chehey, *et al.* describe the Low Water Mark problem as follows:

The example system has at least one data object and three operations: READ, WRITE, and RESET. The operations are used by several processes having various fixed security levels. The system is required to satisfy the simple security and *-property. For simplicity the security levels are assumed to be linearly ordered.

...The low water mark idea is that the data object has a security level that can decrease but not increase except via RESET. A decrease in level occurs when the object is (totally) rewritten by a lower level process. The new level of the object is the level of the calling process.

The Bell and LaPadula simple security and *-properties²² are following requirements: for a process to read an object the process level must dominate that of the object; to write, the object level must dominate that

of the process. A read involves no change of levels for the object; a write causes the object level to drop to that of the process. Reset causes the object security level to become *system high* and the value of the object to become *undefined*.

Chehey, *et al.* require that the *dominates* relation on levels be a total ordering "for simplicity." However, Rushby²³ has shown that the system described is insecure if the levels are only partially ordered.

NON-INTERFERENCE

The security model for our solution to the low water mark problem is a non-interference model. The notion of non-interference assertions was developed by Goguen and Meseguer^{24, 25} and elaborated upon by Rushby²⁶. It provides a powerful and quite general mechanism for describing security policies. Non-interference has been applied successfully in the proof of certain security properties of the Honeywell Secure Ada Target machine²⁷.

Process p_1 is said to be *non-interfering* with process p_2 (denoted $p_1 \mapsto p_2$) if no instruction issued by p_1 can influence the future output of the system to p_2 . A non-interference security policy is simply a set of assertions which characterize which interferences between processes in a system are prohibited (alternatively a binary relation on processes). This notion can be rendered more amenable to formal treatment by the following observation: for any sequence of operations *seq*,

$$p_1 \mapsto p_2 \equiv \text{View}_{p_2}(\text{seq}) = \text{View}_{p_2}(\text{seq}/p_1)$$

where $\text{View}_p(\text{seq})$ is the complete picture that process p has of the state of the system, and seq/p is the subsequence of *seq* obtained by deleting those instructions executed by p .

Our non-interference policy is a simplification of the DoD MLS policy. Processes have associated levels which are related by a total order. Process p_1 may interfere with process p_2 only if $\text{level}(p_1) \leq \text{level}(p_2)$. That is, the security policy is characterized by the following set of non-interference assertions:

$$\{p_1 \mapsto p_2 : \neg \text{level}(p_1) \leq \text{level}(p_2)\}.$$

We notice that it is only the *levels* of individual processes that are relevant to security. Consequently, to demonstrate that security is maintained in the system it suffices to show, for an arbitrary process p , that no instruction executed on behalf of any process at a higher level can affect p 's view. That is,

$$\text{View}_p(\text{seq}) = \text{View}_p(\text{seq}/\text{level}(p)),$$

where $\text{seq}/\text{level}(p)$ is the instruction sequence purged of all instructions executed on behalf of processes at levels not dominated by $\text{level}(p)$. It is proved in²⁷ that this policy implies both simple security and the *-property. Thus, it satisfies the constraints of the low water mark problem.

THE SPECIFICATIONS

In this section we give the Gypsy and Boyer-Moore specifications of the non-interference policy described in the preceding paragraph. For readability we write all Gypsy code in lower case and all Boyer-Moore code in upper case. Enough elaboration is given here (we hope) to give a clear idea of the nature and details of each of the two specifications. The reader interested in the complete set of definitions and lemmas is referred to²⁸. The Gypsy version is described in detail in²⁹.

The Main Data Types

The notions (data types) of process, object, state, and instruction appear in each specification. To avoid entanglement in syntactic details, we give only an outline of the main data types here, namely state, level, and instruction sequence (and relevant auxiliary types). We refer the reader to²⁸ if more details are desired.

Security States. Let us begin with Gypsy. In the Gypsy spec, an *object* is arbitrary; the *object* type is declared to be *pending*. Similarly, the *process* type is *pending*. However, the types *object_level_map*, *object_value_map*, and *process_values_map* are declared in order to specify a security state with the following (Pascal-like) type declaration:

```

type security_state =
  record (values_read: process_values_map;
         object_level: object_level_map;
         object_value: object_value_map);

```

The three types referred to in this record are declared as mappings: for example, one such declaration is

```

type process_values_map =
  mapping from process to value_sequence;

```

where one declares

```

type value_sequence = sequence of value_type;

```

Thus, the values read by process *p* are obtained by taking the `values_read` component of the state and then applying the resulting mapping to the process *p*:

```

state.values_read[p]

```

The Boyer-Moore type declarations are quite similar, except that there are no "mapping types". Let us begin with *processes*. Thus, for example, the values read by a process are a component of the process rather than the result of applying a function to the process. The following syntax is simply a declaration of a process as a record type with two fields. Thus if `(PROCESSP X)` is true, then `(PROC-NAME X)` equals the value of the first field and `(VALUES-READ X)` equals the value of the second field. Conversely, if `ProcName` and `ValRead` are these two respective values, then `X = (PROCESS ProcName ValRead)`, i.e. `PROCESS` is actually a function which constructs a process from a name and some values-read.

Shell Definition

Add the shell `PROCESS` of two arguments
with recognizer `PROCESSP`
and accessors `PROC-NAME` and `VALUES-READ`

(Remark for readers familiar with the Boyer-Moore logic. The reader may notice that we have omitted declarations of the type restrictions, default values, and bottom object from the following shell definition. In fact these are *none*, `ZERO`, and *none*, respectively. Similar simplifications will be made in the other shell definitions presented below.)

In the Boyer-Moore specification, it was convenient to choose to access the values-read as a function of the *name* of a process rather than of the process itself. An advantage to this approach is that it is absolutely clear that the level of a process does not change during the execution of instructions; however, this is also a disadvantage since the model is less general. At any rate, in the Boyer-Moore case, one reads the values from a process name as follows:

```

(VALUES-READ (GET-PROCESS P-NAME STATE))

```

where `VALUES-READ` (defined above) is an accessor for processes and `GET-PROCESS` is a recursively defined function:

```

(DEFN GET-PROCESS (P-NAME PROCESSES)
  (IF (NOT (LISTP PROCESSES))
      F
      (IF (EQUAL P-NAME
                (PROC-NAME (CAR PROCESSES)))
          (CAR PROCESSES)
          (GET-PROCESS P-NAME
                       (CDR PROCESSES)))))

```

Notice that this function is necessary in the Boyer-Moore version because the Boyer-Moore logic does not support mapping types. That is, functions must be defined in the logic rather than being data objects (such as the object `state.values_read` in Gypsy).

So, we have defined the notion of process for the Boyer-Moore version. The notion of *object* is similar, and we omit details:

Shell Definition

Add the shell `OBJECT` of three arguments
with recognizer `OBJECTP`
and accessors `OBJ-NAME`, `VALUE`, and `OLEVEL`

We may now define a state to be simply a record consisting of a list of processes and a list of objects.

Shell Definition

Add the shell `STATE` of two arguments
with recognizer `STATEP`
and accessors `PROCESSES` and `OBJECTS`

Actually no restriction is made on the components of a state, because of the weakness of the Boyer-Moore type (shell) mechanism. Instead, a predicate `PROPER-STATEP` is defined which restricts the class of "states" for the theorem ultimately proved. Thus the following function (predicate) returns `T` (true) exactly when its argument is a state whose components are respectively a list of processes and a list of states. (Thus the functions `PROCESS-LISTP` and `OBJECT-LISTP` are defined first; however, we omit their straightforward definitions here.)

Definition

```

(PROPER-STATEP STATE)
=
(AND (STATEP STATE)
     (PROCESS-LISTP (PROCESSES STATE))
     (OBJECT-LISTP (OBJECTS STATE)))

```

This kind of treatment of states is necessary because the `sequence` of type constructor of Gypsy (used above in the declaration of `value_sequence`) has no real analogue in the Boyer-Moore logic.

Levels. The notion of *level* is specified in each language as a function (which is left undefined). However, as mentioned above, we chose to have the function depend on the process in the Gypsy version but on the process *name* in the Boyer-Moore version.

```

function process_level (p: process):
  level_type = pending;

```

Declaration

```

(LEVEL PROC-NAME) = [unspecified]

```

There is one other difference in the handling of levels by the two versions. The Gypsy version (which was done first) used the integer ordering functions in order to order the levels, thus treating `level_type` (see above) as though it were the type of integers. This had the advantage of allowing the Gypsy simplifier to contribute its built-in knowledge of integers (and their ordering) to the proof. However, the Boyer-Moore version was undertaken with the goal of allowing an arbitrary total order, `DOMINATES`, with the axioms for a total order included. This was indeed accomplished, and no other axioms were needed except that the "system high" level is the greatest level: `(DOMINATES (SYSTEM-HIGH) L)`. (A similar axiom was added for the Gypsy proof as well.)

Instruction Sequences. The notion of instruction is defined as a record in each language, with fields corresponding roughly to the type (i.e. read, write, or reset), process, object, and value. (Of course, the value field is not necessary for a *read* instruction; it is simply ignored.)

```

type instruction_class = (rd, wrt, rst);

```

```

type instruction =
  record (class: instruction_class;
         p: process;
         o: object;
         v: value_type)

```

Shell Definition

Add the shell `MAKE-INSTRUCTION` of four arguments
with recognizer `INSTRUCTIONP`
and accessors `TYPE`, `I-PROC-NAME`,
`I-OBJ-NAME`, and `I-VALUE`

The notion of instruction sequence is declared as a type in Gypsy but is defined by a recursive function in the Boyer-Moore logic (again, because there is no sequence type constructor in that logic):

```

type instruction_sequence =
  sequence of instruction;

```

Definition

```

(INSTRUCTION-LISTP LST)
=
(IF (NOT (LISTP LST))
    T
    (AND (INSTRUCTIONP (CAR LST))
         (INSTRUCTION-LISTP (CDR LST))))

```

The Main Theorem

Following the simple security and *-properties mentioned earlier, we imagine executing instructions which request reads, writes, and resets, where "illegal" requests are ignored. Thus a process may not read an object at a strictly higher level or write-to (or reset) an object at a strictly lower level. Let us begin by stating the main security theorem in each of the two languages. We will then give definitions of functions used in these statements, including *interpret*, which runs the given instruction sequence on the given state (to return a new state), and *purge*, which removes all instructions whose level exceeds the level of the given process. In the Gypsy version, one considers the equality of the values read by a given process in the following two states: the state obtained after running the original instructions, and the state obtained after running the purged instructions. In the Boyer-Moore version, we have chosen to consider these two processes rather than just the values that they have read, since a process is merely a name together with those values. Either way, the definitions of the *purge* function guarantee security of the system because the purged instructions have no effect on the process's view of the system.

```
lemma system_is_secure
  (inseq: instruction_sequence;
   state: security_state;
   p: process) =
  [interpret (inseq, state).values_read[p] =
   interpret (purge (inseq, process_level (p)),
             state).values_read[p]];
```

```
Lemma (SYSTEM-IS-SECURE).
(IMPLIES
 (AND (PROPER-STATEP ST)
      (INSTRUCTION-LISTP INSTLIST))
 (EQUAL (GET-PROCESS
          P-NAME
          (PROCESSES (INTERPRET INSTLIST ST)))
        (GET-PROCESS
          P-NAME
          (PROCESSES
           (INTERPRET (PURGE INSTLIST
                       (PLEVEL P-NAME))
                     ST))))))
```

The function *interpret* takes a sequence of instructions and an initial state and returns a new state. It is in turn defined in terms of a "single stepper" which interprets a single instruction. Notice that the Gypsy definition of *interpret* recursively decomposes the instruction sequence from the right while the Boyer-Moore definition works from the left. Gypsy syntax supports accessing sequences from either end; Boyer-Moore's LISP-like style strongly favors recursively decomposing lists from the left. The Boyer-Moore version of *interpret* "runs" the given instructions in the reverse of the order in which the Gypsy version "runs" the instructions.

```
function single_step (i: instruction;
                    state: security_state)
:security_state =
begin
  exit result =
  if i.class = rd
  then read (i.p, i.o, state)
  else
  if i.class = wrt
  then write (i.p, i.o, i.v, state)
  else reset (i.p, i.o, state)
  fi
  fi;
  pending
end;
```

```
function interpret
  (inseq: instruction_sequence;
   state: security_state): security_state =
begin
  exit result =
  if inseq = null (instruction_sequence)
  then state
  else single_step
        (last (inseq),
         interpret (nonlast (inseq),
                   state))
  fi;
  pending
end;
```

```
Definition
(SINGLE-STEP INST ST)
=
(IF (GET-OBJECT (I-OBJ-NAME INST)
              (OBJECTS ST))
 (IF (EQUAL (TYPE INST) 'READ)
  (READ (I-PROC-NAME INST)
        (I-OBJ INST ST) ST)
 (IF (EQUAL (TYPE INST) 'WRITE)
  (WRITE (PLEVEL (I-PROC-NAME INST))
         (I-OBJ INST ST)
         (I-VALUE INST)
         ST)
 (RESET (PLEVEL (I-PROC-NAME INST))
        (I-OBJ INST ST) ST))))
 ST)
```

```
Definition
(INTERPRET INSTLIST ST)
=
(IF (NOT (LISTP INSTLIST))
 ST
 (SINGLE-STEP
  (CAR INSTLIST)
  (INTERPRET (CDR INSTLIST) ST)))
```

The auxiliary *read*, *write*, and *reset* functions take a security state together with other appropriate arguments (such as process or its level, object, and value), and return a new state. However, the state is unchanged if the relevant levels are inappropriate. For example, here are the two definitions of *write*. Notice that the Gypsy syntax is richer in that its *with* construct allows a convenient notation for updating specified fields of a record.

```
function write (p: process; o: object;
              v: value_type;
              state: security_state)
: security_state =
begin
  exit result =
  if process_level (p) le
    state.object_level[o]
  then state
    with (.object_value[o] := v;
         .object_level[o] :=
           process_level (p))
  else state
  fi;
  pending
end; {write}
```

Recall below that *PROCESSES* picks out the *PROCESSES* field of the given state, and similarly for *OBJECTS*.

Definition

```
(WRITE LEVEL O V ST)
=
(IF (DOMINATES (OLEVEL O) LEVEL)
  (STATE (PROCESSES ST)
    (REWRITE-OBJECT
      (OBJ-NAME O)
      V LEVEL (OBJECTS ST))))
  ST)
```

where (REWRITE-OBJECT O-NAME V L OBJECTS) returns the result of replacing the value of the object named O-NAME with v and the level with L, in the given list of OBJECTS. (We omit the recursive definition of REWRITE-OBJECT.)

It remains to define *purge*. The "values read" component of the new state contains, for each process P, the sequence of values received by P as a result of the READ instructions executed on its behalf. In our model, this is the only information that a process can obtain about the system. Thus, the following are the definitions of *purge*. Notice that the Boyer-Moore function is defined for a much broader collection of arguments. The type-free nature of the logic and the requirement that all functions be total requires that PURGE be defined on arguments which are intuitively quite different than the intended "argument types."

```
function purge
  (inseq: instruction_sequence;
   l: level_type): instruction_sequence =
begin
  exit result =
    if inseq = null (instruction_sequence)
    then null (instruction_sequence)
    else if l ge
      process_level (last (inseq).p)
      then purge (nonlast (inseq), l)
        <: last (inseq)
      else purge (nonlast (inseq), l)
        fi
    fi;
end; {purge}
```

Definition

```
(PURGE INSTLIST LEVEL)
=
(IF (NOT (LISTP INSTLIST))
  INSTLIST
  (IF (DOMINATES
      LEVEL
      (PLEVEL
        (I-PROC-NAME (CAR INSTLIST))))
    (CONS (CAR INSTLIST)
      (PURGE (CDR INSTLIST) LEVEL))
    (PURGE (CDR INSTLIST) LEVEL)))
```

OUTLINES OF THE PROOFS

The main lemmas for the proofs are similar. In each case, the idea is to proceed by some kind of induction on the length of the instruction list. However, in order to obtain a sufficiently strong inductive hypothesis it is desirable to prove a somewhat stronger result than the main theorem (which was called "System Is Secure" above), from which the main theorem follows immediately. The stronger result says that the given process has the same view of the two relevant states, namely the ones obtained with and without running the purged instructions.

```
function process_view_identical
  (p: process;
   state1, state2: security_state)
: boolean =
begin
  exit result =
    ( state1.values_read[p] =
      state2.values_read[p]
      & (all o1: object,
        o1 in could_read (p, state1)
        iff o1 in could_read (p, state2))
      & (all o2: object,
        o2 in could_read (p, state1)
        -> state1.object_level[o2]
          = state2.object_level[o2])
```

```
      & state1.object_value[o2]
        = state2.object_value[o2]));
end; {process_view_identical}
```

where the function *could_read* returns the set of objects that can be read by the given process:

```
function could_read
  (p: process;
   state: security_state): object_set =
begin
  exit (all o: object,
        o in result
        iff process_level (p) ge
          state.object_level[o]);
end; {could_read}
```

```
lemma purge_preserves_process_view
  (inseq: instruction_sequence;
   state: security_state;
   p: process) =
process_view_identical
  (p, interpret (inseq, state),
  interpret
    (purge (inseq, process_level (p)),
     state));
```

And now the Boyer-Moore version:

Definition

```
(PROCESS-VIEW-IDENTICAL P-NAME ST1 ST2)
=
(AND (EQUAL
  (GET-PROCESS P-NAME (PROCESSES ST1))
  (GET-PROCESS P-NAME (PROCESSES ST2)))
  (OBJECT-NAMES-AGREE (OBJECTS ST1)
    (OBJECTS ST2))
  (OBJECTS-MATCH-BELOW-LEVEL
    (PLEVEL P-NAME)
    (OBJECTS ST1)
    (OBJECTS ST2)))
```

where OBJECT-NAMES-AGREE returns T (true) when the two object lists have the same names (in the same order) and OBJECTS-MATCH-BELOW-LEVEL implies that the given object lists agree when restricted to objects below the given level:

Definition

```
(OBJECTS-MATCH-BELOW-LEVEL LEVEL OBJ1 OBJ2)
=
(IF (AND (LISTP OBJ1) (LISTP OBJ2))
  (IF (OR (DOMINATES LEVEL
    (OLEVEL (CAR OBJ1)))
    (DOMINATES LEVEL
    (OLEVEL (CAR OBJ2))))
    (AND (EQUAL (CAR OBJ1) (CAR OBJ2))
      (OBJECTS-MATCH-BELOW-LEVEL
        LEVEL (CDR OBJ1) (CDR OBJ2)))
    (OBJECTS-MATCH-BELOW-LEVEL
      LEVEL (CDR OBJ1) (CDR OBJ2)))
  (AND (NOT (LISTP OBJ1))
    (NOT (LISTP OBJ2))))
```

Thus we are brought to the Boyer-Moore version of the main lemma:

Lemma (PURGE-PRESERVES-PROCESS-VIEW).

```
(IMPLIES
  (AND (PROPER-STATEP ST)
    (INSTRUCTION-LISTP INSTLIST))
  (PROCESS-VIEW-IDENTICAL
    P-NAME
    (INTERPRET INSTLIST ST)
    (INTERPRET (PURGE INSTLIST
      (PLEVEL P-NAME))
      ST)))
```

In both versions, there are two main sublemmas used for proving the inductive step of this lemma, i.e. for proving (roughly) that the lemma remains true when one considers one more instruction on the given list of instructions. The first lemma treats the case that the added instruction has a level which is higher than that of the given process (and may therefore be purged):

```

lemma
  purgeable_instruction_preserves_process_view
  (a: instruction;
   p: process;
   state1, state2: security_state) =
  not process_level (a.p) le
    process_level (p)
  & process_view_identical
    (p, state1, state2)
  -> process_view_identical
    (p, single_step (a, state1), state2);

```

```

Lemma
(PURGEABLE-INSTR-PRESERVES-PROCESS-VIEW).
(IMPLIES
  (AND (PROPER-STATEP ST1)
        (PROPER-STATEP ST2)
        (INSTRUCTIONP INST)
        (PROCESS-VIEW-IDENTICAL P-NAME ST1 ST2)
        (NOT (DOMINATES
              (PLEVEL P-NAME)
              (PLEVEL (I-PROC-NAME INST))))))
  (PROCESS-VIEW-IDENTICAL
   P-NAME
   (SINGLE-STEP INST ST1)
   ST2))

```

The other lemma treats the other case, i.e. where the added instruction is not purged:

```

lemma
  nonpurgeable_single_step_preserves_process_view
  (p: process;
   a: instruction;
   state1, state2: security_state) =
  process_level (a.p) le process_level (p)
  & process_view_identical (p, state1, state2)
  -> process_view_identical
    (p, single_step (a, state1),
     single_step (a, state2));

```

```

Lemma
(NONPURGEABLE-INSTRUCTION-PRESERVES-
PROCESS-VIEW)
(IMPLIES
  (AND (PROPER-STATEP ST1)
        (PROPER-STATEP ST2)
        (PROCESS-VIEW-IDENTICAL P-NAME ST1 ST2)
        (DOMINATES
         (PLEVEL P-NAME)
         (PLEVEL (I-PROC-NAME INST))))
  (PROCESS-VIEW-IDENTICAL
   P-NAME
   (SINGLE-STEP INST ST1)
   (SINGLE-STEP INST ST2)))

```

Both proofs use a number of additional subsidiary definitions and lemmas. However, the Boyer-Moore system certainly provides a much more powerful level of automatic support.

COMPARING THE TWO METHODOLOGIES

We compare the following aspects of the Gypsy and Boyer-Moore methodologies:

- Specification style
- Proof management
- Style of interaction with the prover
- Soundness

Specification Style

Gypsy is a rich language in that it has sets and functions as first-class data objects, first-order quantifiers, and an expressive typing discipline for user-defined types. The Boyer-Moore logic does not have these features, but its solid treatment of recursion and lists, along with its capability for introducing data types with the so-called shell principle, allows one sufficient specification power. Gypsy also is richer in that it has procedural constructs, though for high-level specs (such as the Low Water Mark example) users of Gypsy have found it advantageous to use a functional style. In spite of the different notions of data type and the

greater expressive power of the Gypsy language, the specs are clearly quite similar for the two versions of the Low Water Mark example that are presented here.

The issue of types deserves further comment. Consider the following (incorrect) statement of the main theorem in the Boyer-Moore logic. Sadly, an earlier version of our specification contained this misstatement. The reader is invited to find the error before reading further.

```

Lemma (SYSTEM-IS-SECURE).
(IMPLIES
  (AND (PROPER-STATEP ST)
        (INSTRUCTION-LISTP INSTLIST))
  (EQUAL
   (GET-PROCESS P-NAME
    (INTERPRET INSTLIST ST))
   (GET-PROCESS P-NAME
    (INTERPRET
     (PURGE INSTLIST
      (PLEVEL P-NAME))
     ST))))

```

The problem with this statement is that GET-PROCESS is defined so that its second argument is (expected to be) a list of processes, not a state. In fact, the two sides of the equality above are actually both provably equal to F (false), under the given hypotheses! An analogous error in the Gypsy text would have been caught by the type-checker. However, the problem of matching formal specifications to intuitive requirements remains a central issue in program verification research.

Proof Management

The Gypsy system allows one to defer the proofs of lemmas during a proof session. This capability is amenable to a top-down proof style which is quite natural. The Boyer-Moore system allows one to add axioms, which enables one to have the same top-down capability to some extent; one simply assumes seemingly necessary lemmas before proving the main result, and then one goes back and proves those supporting facts. However, that strategy is awkward with the Boyer-Moore system since event histories are totally ordered.

Style of Interaction with the Prover

The Boyer-Moore prover is much more powerful than is the Gypsy prover, and thus allows much larger proof steps and is significantly less tedious to operate. Even though the two verifications discussed here each contain about thirty lemmas, the Boyer-Moore prover proved each of those lemmas automatically (occasionally with some simple hints supplied with the statements of the lemmas), while the Gypsy prover required considerable tedious interaction in order to prove many of the lemmas. However, the powerful heuristics and rule-based rewriting capabilities of the Boyer-Moore prover also make its behavior somewhat unpredictable and also quite difficult and frustrating to control at times, though it prints out useful information to help discover what additional lemmas are needed. The level of interaction is not the only significant difference in the style of interaction. It is much easier with the Boyer-Moore system to modify an existing proof and replay the resulting definition and proof commands. But as mentioned above, the Gypsy system is much more flexible about the order in which one gives proofs.

Soundness

The Boyer-Moore logic, as described completely in²⁰ and Chapter 3 of¹⁹, has simple and well-understood operational and denotational semantics in which every proved theorem is in fact true. Unfortunately, the same cannot be said of Gypsy. Moreover, the Gypsy system does not have a mechanism for ensuring that all lemmas have been proved, nor does it guarantee that circular arguments (in which lemmas use each other for their proofs) are not constructed. Finally, there is empirical evidence over 15 years for virtually bug-free performance of the Boyer-Moore implementation that has not been matched by the Gypsy implementation.

CONCLUSIONS

Despite certain shortcomings, we believe that the Boyer-Moore logic provides a reasonable specification alternative for secure systems, particularly at the model level. Soundness of the logic and the care with

which it is implemented in the theorem prover are strong advantages of the Boyer-Moore system over Gypsy or other currently available verification systems. Gypsy, on the other hand, is an expressive and versatile language which provides the benefits of data types, data abstraction, concurrency, conditional handling, procedural semantics, etc.

We believe that the relative strengths of the Gypsy and Boyer-Moore systems are in fact quite compatible. Work is already underway to remedy some of the soundness deficiencies of Gypsy, and a preliminary system for the Boyer-Moore logic has been constructed³⁰ that uses the Boyer-Moore prover as a component but also allows more user control. Moreover, a form of quantification has already been added to the Boyer-Moore logic and prover³¹ and plans are under way to add sets, full first-order quantification, and more flexible structuring of proofs. Preliminary investigation has also begun into developing a Gypsy-like syntax for the Boyer-Moore logic. This "merger" of Gypsy and the Boyer-Moore systems is the subject of an active research effort at Computational Logic, Inc., with the intended result to be called *Rose*. The primary component omitted from *Rose* is expected to be the procedural part of Gypsy. The hope is that technology will continue to be developed toward obtaining efficient implementations of functional languages, particularly with respect to seeming opportunities for concurrent evaluation.

Our experiences to date suggest that a fundamental requirement for any successful verification is that the person(s) doing the verification understand the theorems to be proved. A system with Gypsy's flexibility of language and use and with the Boyer-Moore system's proof power and clear semantics would be a great aid in this respect.

ACKNOWLEDGEMENTS

We thank Dave Plummer of the University of Texas for helpful comments. We also thank Tom Haigh of Honeywell Secure Computing Technology Center for co-development of some of the ideas on non-interference (cf.²⁷). Finally, we thank our colleagues at the Institute for Computing Science at UT and Computational Logic Inc. for a stimulating environment.

References

1. D.I. Good, R.L. Akers, L.M. Smith, "Report on Gypsy 2.05", Tech. report ICSCA-CMP-48, Institute for Computer Science and Computing Applications, The University of Texas at Austin, February 1986.
2. D.I. Good, B.L. Divito, M.K. Smith, "Using The Gypsy Methodology", Tech. report, Institute for Computing Science, University of Texas at Austin, June 1984.
3. Department of Defense, "*Trusted Computer Systems Evaluation Criteria*", DOD 5200.28-STD, December, 1985.
4. M.K. Smith, A. Siebert, B. Divito, and D. Good, "A Verified Encrypted Packet Interface", *Software Engineering Notes*, Vol. 6, No. 3, July 1981.
5. D.I. Good, A.E. Siebert, L.M. Smith, "Message Flow Modulator Final Report", Tech. report ICSCA-CMP-34, Institute for Computing Science, University of Texas at Austin, December 1982.
6. D.I. Good, "SCOMP Trusted Processes", ICSCA Internal Note 138, The University of Texas at Austin.
7. W.E. Boebert, W.D. Young, R.Y. Kain, S.A. Hansohn, "Secure ADA Target: Issues, System Design, and Verification", *Proc. Symposium on Security and Privacy*, IEEE, 1985.
8. J. Keeton-Williams, S.R. Ames, B.A. Hartman, and R.C. Tyler, "Verification of the ACCAT-Guard Downgrade Trusted Process", Tech. report NTR-8463, The Mitre Corporation, 1982.
9. K.N. Levitt, L. Robinson, B.A. Silverberg, "The HDM Handbook," vols. 1-3", Tech. report, SRI International, June 1979.
10. E.J. McCauley and P.J. Drongowski, "KSOS: The Design of a Secure Operating System", *Proceedings of the AFIPS Conf., Vol. 48*, AFIPS Press, Arlington, VA., 1979.
11. "System Specification for SAC Digital Network", ESD-MCV-1A, ITT Defense Communications Division, Nutley, N.J..
12. Warren A. Hunt, "FM8501: A Verified Microprocessor", Tech. report ICSCA-CMP-47, Institute for Computing Science, University of Texas at Austin, December 1985.
13. W.R. Bevier, W.A. Hunt, W.D. Young, "Toward Verified Execution Environments", *Proceedings of the 1987 Symposium on Security and Privacy*, IEEE, 1987.
14. M. Chehey, M. Gasser, G. Huff, J. Millen, "Verifying Security", *ACM Computing Surveys*, Vol. 13, No. 3, September 1981, pp. 279-340.
15. Richard Kemmerer, "Verification Assessment Study Final Report", In 5 volumes, unpublished.
16. Michael K. Smith, "Low-Water-Mark, Gypsy Style", Internal Note 159, Institute for Computing Science, The University of Texas at Austin, February, 1985.
Michael K. Smith, "Low-Water-Mark Using Abstract Types", Internal Note 158, Institute for Computing Science, The University of Texas at Austin, February, 1985.
18. J. McCarthy, et. al., *LISP 1.5 Programmer's Manual*, MIT Press, Cambridge, MA., 1965.
19. Robert S. Boyer and J Strother Moore, *A Computational Logic*, Academic Press, New York, 1979.
20. R.S. Boyer and J S. Moore, *Metafunctions: Proving Them Correct and Using Them Efficiently as New Proof Procedures*, Academic Press, 1981, pp. 103-185.
21. Richard Cohen, "Proving Gypsy Programs", Tech. report ICSCA-CMP-51, Institute for Computing Science, University of Texas at Austin, 1986.
22. D.E. Bell and L.J. LaPadula, "'Secure Computer System: Unified Exposition and Multics Interpretation'", Tech. report MTR-2997, MITRE Corp., July 1975.
23. John Rushby, "'The Low-Water Mark Example'", in Richard Kemmerer, editor, *Verification Assessment Study Final Report*, Vol. 5, unpublished.
24. J.A. Goguen and J. Meseguer, "Security Policy and Security Models", *Proc. Symposium on Security and Privacy*, IEEE, 1982, pp. 11-20.
25. J.A. Goguen and J. Meseguer, "Unwinding and Inference Control", *Proc. Symposium on Security and Privacy*, IEEE, 1984, pp. 75-86.
26. John Rushby, "Mathematical Foundations of the MLS Tool for Revised Special", Draft internal note, Computer Science Laboratory, SRI International, Menlo Park, California.
27. J.T. Haigh, W.D. Young, "Extending the Non-Interference Version of MLS for SAT", *Proceedings of the 1986 Symposium on Security and Privacy*, IEEE, 1986, pp. 232-239.
28. Matt Kaufmann and William D. Young, "Comparing Gypsy and the Boyer-Moore Logic for Specifying Secure Systems", Tech. report, Institute for Computer Science and Computing Applications, The University of Texas at Austin, 1987.
29. William D. Young, "The Low-Water-Mark Problem Using Non-Interference", Internal note, Honeywell Secure Computing Technology Center, April, 1986.
30. Matt Kaufmann, "A Primitive User's Manual for an Interactive Version of the Boyer-Moore Theorem-Prover (Draft)", ICSCA Internal Notes 234 (Part 1) and 235 (Part 2), The University of Texas at Austin.
31. R.S. Boyer and J S. Moore, "The Addition of Bounded Quantification and Partial Functions to the Boyer-Moore Logic and Theorem Prover", Tech. report ICSCA-CMP-52, Institute for Computer Science and Computing Applications, The University of Texas at Austin, January 1987.

LOCKING COMPUTERS SECURELY

O. Sami Saydjari, Joseph M. Beckman, Jeffrey R. Leaman

Office of Research and Development
National Computer Security Center

ABSTRACT

Progress has been slow over the last 15 years in the relatively new field of computer security. Every initiative started from scratch to develop a secure computer. First prototypes, built in software, were slow and difficult to use. LOCK is a technology research and development project to build a hardware-based Reference Monitor module. This module will be generic and thus reusable on many different computers. Full advantage will be taken of inexpensive generic cryptographic modules currently in development.

I. HISTORY AND SUMMARY

There is an immediate need for computing facilities to handle data at different security levels for users possessing different levels of clearance. These multilevel secure computers will fulfill three major requirements.

First, secure computers must meet the need for inherently multilevel applications processing different levels of classified information and reporting to users cleared to different levels. For example, Military Air Command maintains flight schedule information. Most of the information is unclassified except for parts having to do with covert missions. The intelligence community's solution of clearing everyone to the highest level is impractical given the number of personnel requiring access to this database. This problem will only get worse as the information age forces us to integrate more and more information processing systems.

Second, enabling computers to serve users with different clearances avoids the need to maintain duplicate computers for different classification levels. The potential money savings is significant. The alternative of clearing all users to the highest level is too expensive, impractical, and unacceptable from a security standpoint.

Finally, secure computers are extremely important even for those systems not requiring multilevel applications. For example, there is a significant threat from malicious software introduced onto the system from a multitude of possible routes. [MYER80] Such malicious software could destroy invaluable information, spoof users into taking inappropriate action, or prevent computers from operating at critical times.

The need for secure computing in both defense and industry is reaching critical proportions and will only grow. The Logical Coprocessing Kernel (LOCK) project promises to solve a substantial part of this problem.

Current systems, and most of those under development, attempt to provide multilevel security in software by redesigning the operating system. The purely software approach has four serious disadvantages compared to the primarily hardware approach used in LOCK:

1. DECREASED ASSURANCE
since software malfunction could cause total security failure,

2. DECREASED PERFORMANCE
to usually unacceptable levels because of the high overhead from the security access checks done in software,

3. LOSS OF EXISTING APPLICATION SOFTWARE because of the extensive redesign of the operating system, and

4. INABILITY TO FUNCTIONALLY ENHANCE the operating system without requiring expensive and time-consuming re-verification and reevaluation.

The LOCK hardware-oriented approach promises high assurance and reasonable performance derived from the implementation of a physically separate and parallel security-enforcing module called the system-independent, domain-enforcing, assured, reference monitor (SIDEARM). Furthermore, because the security-related functionality is in the SIDEARM, the software operating system is not security-relevant and, therefore, requires no re-verification when the operating system software is updated. This approach will allow the preservation of most of the application software programs written for that operating system and its subsequent releases.

The LOCK program has at least a 12-year history. The project sprung out of the Provably Secure Operating System [NEUM77] study, begun in 1975 at the Stanford Research Institute. An implementation of the study's recommendations by Ford Aerospace [FORD81] began in 1980. The project goals proved to be too ambitious and the resources allocated to reach those goals were too limited. The hardware part of the Trusted Computing Base (TCB) under this project was continued under the new name Secure Ada Target (SAT) in 1982

by Honeywell. This marked the beginning of the current three phase development described below.

LOCK is the third phase of a continuing project previously called SAT. The SAT project, begun in 1982, was a research effort to design secure computers. Phase one (SAT-0) yielded the high level requirements specification in 1983. [HONE83] Phase two (SAT-I) yielded the intermediate design specification in 1986. [HONE86] Phase three (SAT-II), later renamed LOCK, will yield a detailed design specification and a secure microcomputer prototype by 1990.

II. TECHNICAL APPROACH

The LOCK project goal is to develop a hardware-oriented solution to the computer security problem of providing multilevel security (MLS) for general computers. (MLS provides the ability to process different levels of classified information so that only properly-cleared users may access it.) The heart of the solution is the separate security-enforcing module called SIDEARM. The SIDEARM will be designed, built, and then integrated into an existing microcomputer. The security enforcement will be highly assured to allow certification at the highest level (A1) defined by the Department of Defense Trusted Computer System Evaluation Criteria. [TCSE85]

A. Project Goals

LOCK is a very ambitious information security technology development project. We wish to provide a foundation for current and future secure information systems. This paper establishes the goals of both the base technology and its extensions to address a myriad of computer and communications security problems. LOCK provides the basis for the solutions. We count on industry and other projects to extend and apply this base to reap the full benefit of this technology.

The ultimate goal is to produce the SIDEARM as a standard product to retrofit into most existing computers and included or offered as an option in new computers, like a coprocessor.

LOCK strives to produce very secure computers without severely impacting performance. This work should set standards for a new rating, possibly A2; however, the minimum acceptable level will be A1. While meeting the requirements for at least A1, the functionality of the LOCK is desired to be equal to that of the unmodified base computer. Although this is very feasible, certain security-related limitations may be necessary to meet the assurance requirements.

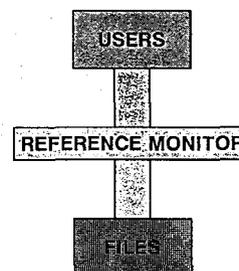
Preserving performance of the machine is another major concern. The design team is striving to achieve 90% of the speed of specified benchmarks when compared to the unmodified computer. The performance area is one that has severely affected previous computer security projects. With the hardware approach of the security module,

the minimum acceptable level is 80%.

B. Reference Monitor

The reference monitor model (see Figure a) is inherent to the design of secure computers. In the model, the reference monitor acts as a guard between people and information. There are three properties the reference monitor must possess. The ideal "guard at a key desk" analogy best explains how this model works. The first property is that the reference monitor must always be invoked; (1) the guard must always be on duty, and there must be no way to obtain a key without being confronted by him. Second, the reference monitor must be verified to operate correctly; (2) the guard must be responsible for doing his job correctly. He knows that one has to have an identification badge and must be on the key access list. Finally, the reference monitor must be tamperproof; (3) there must be no way to hinder or affect the proper operation of the guard. One cannot get the guard confused or substitute a guard of one's own choosing.

REFERENCE MONITOR CONCEPT



A REFERENCE MONITOR MUST BE

1. ALWAYS INVOKED
2. VERIFIED CORRECT
3. TAMPERPROOF

Figure a

C. Alternative Solutions

There are essentially two ways to implement the reference monitor model. Previously, the method of choice was to implement in software, thereby attacking the computer security problem at the operating system level. This approach has met with four serious problems. LOCK is a hardware-based approach that attacks the security problem at the machine level. Both implementation methods are described below.

1. Software

The traditional approach implements the security kernel in software, resulting in a poor instantiation of the reference monitor model. At system boot-up, all of the security-relevant information, tables, etc., need to be loaded from memory into the reference monitor. To do this, the reference monitor has to be bypassed, thereby violating one of the reference monitor properties.

To verify that this type of system is correct, one must explicitly show that there is no way for the user to hack up the security tables or interfere with the security mechanisms that are in place. It becomes a significant task to show that of all possible programs, no program surreptitiously modifies any of this information. Since the Central Processing Unit (CPU) and memory resources are shared between the user and TCB, there is ample opportunity for tampering to take place due to the lack of physical separation.

Performance has been the high price paid for software security kernels. The cause of performance degradation is the multiplexing of resources between the operating system and the reference monitor and increased overhead associated with frequent context switching. The same CPU handles all of the security-related processing in addition to the normal processing of the system in a software implementation. Memory resources are also shared in a software implementation. A portion is allocated for security-relevant information, and the rest is used in the normal course of processing. This dramatically increases the load placed on a single set of resources and decreases the assurance because of the intermixing of computational and security-related information. A context switch, the complete replacement of processed information in the CPU required by a change in subjects, is also a tremendous drain on the processing capability of the computer. These factors have degraded the performance of secure software implementations to as low as 10% of the unmodified operating system. [GOLD84] Additionally, there is some question about the level of assurance gained since some applications, like database management systems (DBMS), need to reach directly into the machine level -- completely bypassing the security mechanisms.

As mentioned earlier, the software approach involves a major redesign or restructuring of the operating system kernel. Many existing application programs require certain services from the operating system. The entry points for these services often must be redefined when the kernel undergoes such significant modification. The loss of these entry points severely limits the compatibility of the operating system with existing applications.

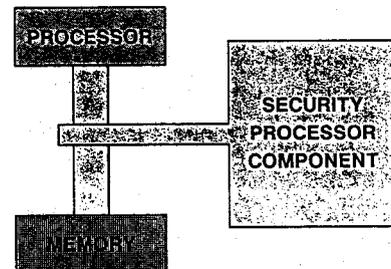
Enhancing an operating system's functionality usually involves making changes to the kernel. Even if the changes are not ostensibly security-relevant, it must be explicitly shown through verification (or revalidation at lower levels of security) that the security of the system has not been affected as a result of subtle interactions between the software modules. Currently, verification of computer systems is expensive and time-consuming. Additionally, no claims about the security of the new operating system may be made until that version of the operating system has been evaluated, a time-consuming procedure. The decision that must be made is whether to undergo another verification and evaluation

or to continue with the existing, outdated operating system.

2. Hardware

The hardware-based approach taken by the LOCK project is a much closer match to the reference monitor model (see Figure b). The problem that arose concerning bypassing the reference monitor is eliminated because the SIDEARM is a separate processor, with its own memory, that is running before the user's processor boots up. In addition, the security mechanisms and tables are self-contained within the SIDEARM, thereby making the system verification easier. Because SIDEARM is a separate resource, there is no physical way for the user to access this memory; it's trivial to verify that the user can't alter the security-relevant information. Finally, a user cannot initiate a process that will tamper with the security-relevant operations because none of SIDEARM's processing resources are under control of the user.

SECURITY PROCESSOR COMPONENT APPROACH



REFERENCE MONITOR CRITERIA

1. ALWAYS INVOKED -- NO WAY TO BYPASS
2. VERIFIED CORRECT -- SIMPLER; MACHINE INDEPENDENT
3. TAMPERPROOF -- NO WAY TO ATTACK SECURITY PROCESSOR COMPONENT

Figure b

The design approach taken by the LOCK project is rigid resource separation (see Figure c). The computational and security resources are segregated at the system design level, and this segregation is carried down to the physical implementation. This approach yields two significant benefits. First, unbyassibility is guaranteed by SIDEARM having exclusive possession of object-addressing information and exclusive control over the memory management unit (MMU) (see paragraph E.4.). Second, the physical separation prevents any tampering on the part of the user.

LOCK will initially be a set of boards, but our goal is to reduce this down to a chip or set of chips by using either very large scale integration (VLSI) or very high speed integrated circuitry (VHSIC) technology.

D. System-Independent, Domain-Enforcing, Assured Reference Monitor (SIDEARM)

SIDEARM is the hardware instantiation of the reference monitor and is

the heart of the LOCK technology development effort. SIDEARM is a separate embedded computer, with its own processors and memory, that controls the resources of the host computer by mediating all accesses to those resources by users operating on the host CPU (see Figure d).

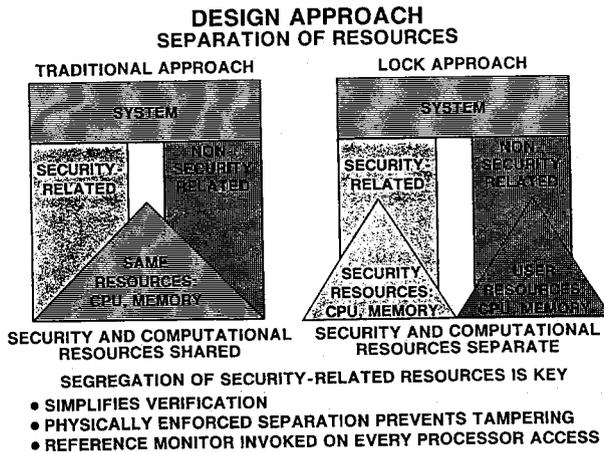


Figure c

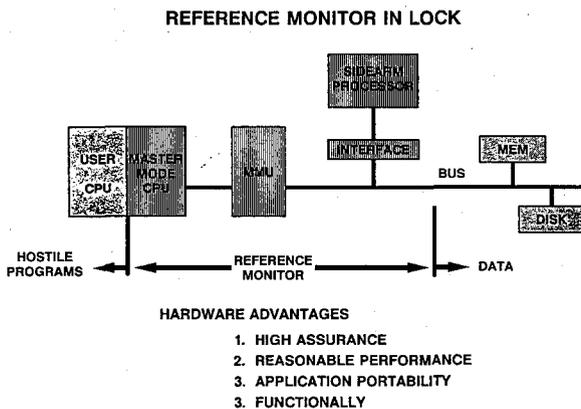


Figure d

Our goal is to develop a generic SIDEARM module that is independent of the computer system it monitors. This does not mean that it will be a black box that one magically affixes to the cabinet of the host computer, making it automatically secure. Rather, the intent is to minimize the replication of work when securing different computers and operating systems by capturing the essence of the TCB - the reference monitor - in a generic module. The machine-specific requirements involving the connection of the SIDEARM module to a particular computer are reasonably small and modularized to interfaces that can be customized.

SIDEARM has three important properties making it generic: (1) it manages the identification and security labeling of all objects and subjects; (2) it implements the mandatory security policy based on these

identifications and security attributes; and (3) it is guaranteed not to be bypassed because it will be physically impossible for the CPU to address its own memory without going through the SIDEARM to get the object's address. These databases and operations are all independent of the host system.

The security databases and operations are not only generic, but they allow for a very flexible and powerful security policy. For example, the dominance relationship which determines access between subjects and objects [BELL75] will be implemented as an explicit, partial ordering data structure. This means that the security lattice (actually, Partially Ordered Set or POSet) can be dynamic, limited to points in the lattice that are truly needed, and have multiple roots, thereby getting rid of the dangerous concept of "system high." [FERG86]

The purpose of developing the SIDEARM module is to provide vendors with a foundation for computer security that they can use to minimize the cost and time required to secure their own computers. No longer will the vendors have to become Criteria lawyers to interpret each and every Criteria requirement for their system. SIDEARM will be precertified to meet a base percentage of the information security requirements. It only remains to demonstrate that the module has been hooked into their computer system correctly and that the remainder of the requirements not met by the SIDEARM module are implemented by the host system.

Providing seed money for the initial development is our way of encouraging industry to both produce these devices and use them in securing their own products. Within LOCK, a SIDEARM module will be retrofitted to an existing computer as a proof-of-principle of the generic nature of the module and as a worked example of how a retrofit is done. This information will be made available to vendors wishing to retrofit their own computers using SIDEARM modules.

Finally, we will publish the documentation on the interface to the SIDEARM module, and vendors designing their next-generation computers can incorporate the capability to insert the SIDEARM module into their new systems without the expense of retrofit changes. In summary, SIDEARM is intended to secure the nation's computer systems cheaply and efficiently at a very high level of assurance.

E. The Architecture

Honeywell's XPS 100/20 computer will be the platform for the LOCK. The XPS 100/20 is an MC68020 microprocessor-based computer with a VME bus architecture, running UNIX System V.

The XPS 100/20 was chosen for two reasons. A study of three different, popular, 32-bit microprocessors [HONE86] found that the MC68020 has a flexible coprocessor interface that makes it easy to adapt to the requirements of LOCK technology.

Furthermore, proprietary data necessary to integrate SIDEARM is readily available since it is a Honeywell machine. Although one of the goals of the LOCK program is to produce a generic computer security module that may be used on many different machines with a minimum of modification, the interface is built for a particular system and requires low-level implementation detail.

1. SIDEARM

SIDEARM looks like a coprocessor or I/O device (depending on how it's retrofitted) to the host system. It contains its own processor (actually between one and four MC68020's), its own VME bus, and its own primary and secondary memory. More processors were added in an effort to lessen the chance that SIDEARM, with its system-enforced access checks, will be the performance bottleneck that has plagued other computers with added security functionality.

Each of the major subsystems in SIDEARM has its own processor, which may be real or virtual. Virtual processors (VP's) communicate by way of a message passing system. The format is the same whether or not the VP is on the same physical processor. This makes it easier to add more physical processors if necessary and to move VP's to different physical processors for performance reasons.

The first major subsystem is a front-end filter which screens out illegal requests from the host. Since this is the only point where the host can communicate with SIDEARM, this subsystem is host specific. Legal host requests are queued until they can be serviced. A complementary VP allows SIDEARM to access the host's resources such as host memory. These requests are coordinated by a resource manager (detailed below).

The main VP is the instruction processor. Its job is to take a host-initiated request, give the appropriate part to the other VP's, and execute the corresponding high-level algorithm.

A resource manager provides the other VP's with access to system resources, including host primary and secondary memory, SIDEARM shared memory, host devices, and host real-time clock. A media manager is responsible for operations affecting the Global Object Table (the data structure that contains all the security-related information needed for access control, resident only in SIDEARM's internal memory) or SIDEARM Resident Objects (other data objects used in security-relevant processing).

The Audit Processor has its own media, a laser disk. If this disk becomes full, the operator will be notified, and audit blocks will be stored directly on SIDEARM's hard disk. If this last disk becomes full, the system will lockup in the interest of security. This two-tier backup of audit data is not called out in the Criteria but represents one area of increased assurance for LOCK.

The last VP is the unique identification (UID) generator. It will produce a unique identifier that is encrypted. Encryption renders the UID "opaque," in the sense that the user-visible UID can not then be used to convey any information.

SIDEARM, as the instantiation of the reference monitor, is responsible for checking access rights and type enforcement controls. Final access rights are the "AND"ed rights from the mandatory access controls (MAC's), the discretionary access controls (DAC's) and the type enforcement controls [SAYD86].

2. SIDEARM Encryption

Device (SED)

The SED is a TEPACHE-based [KIBA86] cryptographic module used for three purposes in LOCK. First, it is used to encrypt SIDEARM media. This complements the bulk encryption device (BED) which encrypts the host's secondary memory.

Second, the SED encrypts the UID attribute of objects to close a covert channel. This covert channel rested on a subject's ability to determine how many objects had been created. By one subject creating either a large or a small number of objects, another subject could monitor that number by creating an object of its own and looking at the UID (assuming UID's are monotonically increasing) and decode information over time. By encrypting the UID's, a subject cannot determine the absolute or relative number of objects created.

Third, the SED is used to manage the cryptographic keys for the BED. The System Security Officer will insert his crypto-ignition key (CIK) into this module to turn on the LOCK. When the CIK is inserted (during normal operation), the host's primary memory has unencrypted information that is potentially sensitive (classified). When the CIK is removed, all memory is nonsensitive (primary has no information and secondary is encrypted) and can be treated as any other high-value piece of office equipment. Cryptographic keys will, themselves, be encrypted within the encryption devices and stored on SIDEARM's secondary memory. The owning cryptographic module will retrieve, decrypt, use, and then reencrypt keys whenever required by the SED or the BED. Neither the host system nor SIDEARM (exclusive of the SED) will ever access or store unencrypted keys.

At some future time, the SED may be used as part of a trusted path between the TCB and the user's terminal. Information could be encrypted at the terminal and sent to SIDEARM. Since only SIDEARM would be able to decrypt and the user's terminal encryption device able to encrypt it, untrusted software would not be able to generate or observe this information.

3. Bulk Encryption Device

(BED)

The bulk encryptor also contains the TEPACHE which is used to encrypt all data stored on the host system disks, tapes, and floppy disks. Furthermore, it will be used to encrypt communications on the network interface port. All of the nonprimary memory devices and the communications port are located on their own bus, separated from the CPU/main memory/SIDEARM bus by the bulk encryptor (see Figure f). Secondary memory may be treated as nonsensitive (unclassified) and no longer must be physically secured when unattended.

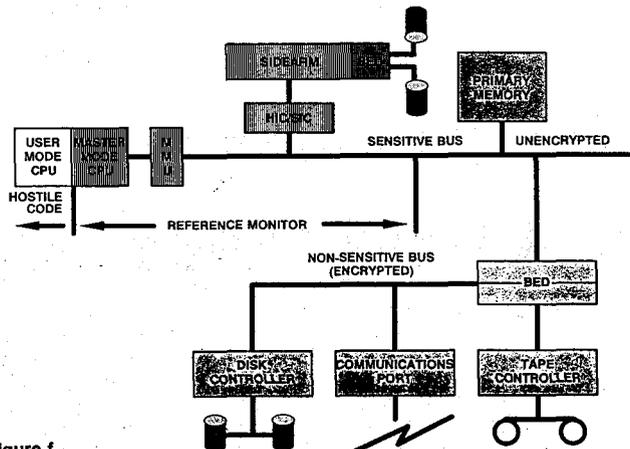


Figure f

Turning secondary memory nonsensitive (unclassified) has another important result. Previously, the device controllers either had to be verified, trusted, or front-ended with antilisting logic to prevent the reception of unencrypted information not intended for that particular device. Now this, too, is no longer necessary. Commercial, off-the-shelf products can be used on this bus, and there is no need to verify their trustworthiness.

Two factors argue against designing the LOCK with a single encryption device. The first is performance. Splitting the functionality between two encryption devices minimizes performance degradation. The second reason is that by having a separate bulk encryptor with minimal logic surrounding it, the sensitive/nonsensitive (red/black) interface is physical. If the SED were used to encrypt the host's secondary memory of communications port, one would have to rely on the system working correctly (and prove this) to ensure that no sensitive (unencrypted) data were ever written to the nonsensitive bus. Placing an encryption device in-line allows us assurance (without incurring the cost of verifying the device controller's software) that sensitive data will never be mishandled.

4. Memory Management Unit

(MMU)

The MMU for the LOCK is the commodity Motorola 68851 MMU. The host CPU queries SIDEARM when a subject first makes a request for an object. The

appropriate information (the global object table, kept on SIDEARM's hard disk) is checked and access is permitted or denied, based on DAC, MAC, and type enforcement constraints. SIDEARM then loads the information into the MMU. The MMU caches the access rights returned until a context switch forces the flushing of the cache. This context switch happens when there is a subject change on the system or when a user effects a change in the level he is processing.

To increase security, customized enhancements to the MMU are being included in the design. Certain master mode code (also called supervisor state or ring-0 code) will be kept in protected PROM's on the MMU, addressable only when the machine is in master mode. Examples include portions of the interrupt handler, fault handler, and the subject manager.

5. Host-SIDEARM Interface

The host and SIDEARM communicate through a custom interface device called the host interface controller (see paragraph E.1., also called the VP front-end filter). This device encapsulates the specific electrical and protocol requirements of the host computer bus and acts as a driver to relay CPU requests to SIDEARM and to receive the responses. The SIDEARM interface controller is SIDEARM's equivalent mechanism for receiving the requests and then sending signals back to the host.

F. Interdependence of COMPUSEC and COMSEC

LOCK contains a cryptographic subsystem composed of two distinct devices: (1) the SIDEARM encryption device (SED) and (2) the bulk encryption device (BED). This subsystem is also called the communications security (COMSEC) subsystem, but in actuality, secure communication outside the system is only one of its functions.

The embedding of COMSEC into LOCK was made possible by a major advance by the Development Center for Embedded Cryptographic Products in the development of generic low-cost COMSEC modules. The purpose and benefit of these modules for COMSEC is the same as those the SIDEARM module will have for computer security (COMPUSEC).

The COMSEC and COMPUSEC are interdependent in LOCK. This means that a subset of the security requirements for each is attained by the use of the other. In the lingo, this makes LOCK an information security (INFOSEC = COMSEC + COMPUSEC) development.

The COMPUSEC depends on the COMSEC for memory encryption to increase assurance, UID encryption to close a covert channel (see paragraph E.2.), and external network encryption to secure information leaving the computer system. The COMSEC depends on the COMPUSEC to meet certain requirements involving the control of the COMSEC and to

provide a secure environment within which the COMSEC can operate. Both the COMPUSEC and COMSEC systems in LOCK are critical to meeting the complete security requirements of information in a general system of computing devices.

G. Software

Although essentially a hardware-based approach, LOCK is not without software. The software is required for two reasons: (1) to allow flexibility during prototyping and (2) to accommodate mutable and system-specific security requirements for a particular computer type and at particular computer sites. The first requirement is reduced for final production machines, but maintainability will require that some portions remain in firmware (e.g. physically-protected ROM). Much of the generic functions can be implemented in hardware.

There are four major blocks of software in LOCK: SIDEARM, kernel interface software (KIS), kernel extensions (KE's), and the host operating system (UNIX) (see Figure g). Each block is discussed in detail below in terms of function, content, interfaces to other blocks, and verification considerations.

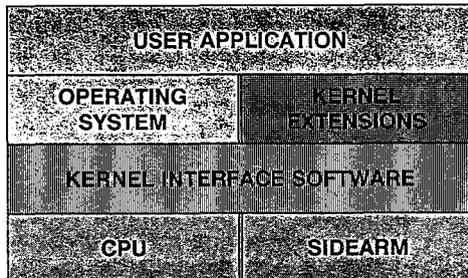


Figure g

1. SIDEARM

The SIDEARM software implements the LOCK reference monitor. The SIDEARM has a set of externally-visible operations plus internal software and databases necessary to perform its task. The exact specification of the visible operations and their parameters will be specified in an Interface Control Document to be released toward the end of 1988.

SIDEARM will contain several thousand lines of high order language software. The reference monitor function of SIDEARM is implemented by several major software subsystems managing the resources of SIDEARM and implementing the instruction requests from the host (see paragraph E.1.).

The host CPU interfaces to the SIDEARM software via the KIS. The security-relevant operations executed by the

host are passed through the KIS, over the host bus to SIDEARM. The SIDEARM then queues and fulfills the request if it is allowed by the security policy.

The verification of the SIDEARM software must be extremely rigorous. SIDEARM is intended to be a generic device that is designed once. All properties required of the reference monitor (including simple-security, the *-property, type enforcement [see paragraph I.], and conditional-non-interference [see paragraph H.2.]) must be shown to hold for all SIDEARM software. This is a very expensive and time consuming process, but it is worthwhile for AI assurance and the cost and time impact is minimized since SIDEARM is only designed and verified once.

2. Kernel Interface

Software (KIS)

The KIS is essentially the driver software for the SIDEARM device on the bus. The host CPU makes security-related requests of the SIDEARM device via the KIS. Normally, device drivers are implemented directly by the host operating system. Because this software must operate correctly for the TCB to function properly, this particular device driver must be segregated from the operating system and verified to function correctly and be unby-passable. In simple terms, the KIS is the connective software that allows the host CPU to communicate with the SIDEARM.

The KIS is intended to be small and minimally privileged. Some of the functions will have to operate in master mode, but they will only have restricted access to the TCB. Innate security-relevant operations designed into the host CPU, such as interrupt handling, will have to be performed by the KIS.

The KIS is an intermediary between the host operating system running on a particular host CPU and the generic SIDEARM. As such, the part of the KIS interfacing with SIDEARM will be highly machine-dependent and will have to be customized when the SIDEARM is ported to different machines. The resource management portion of the KIS, on the other hand, should be fairly general and should require minimum rework when ported. Furthermore, the interface to the KIS should remain stable across different computers and within the same computer when porting to a different operating system.

The verification of the KIS should be somewhat simpler than that of the SIDEARM. The verification is functionally-oriented as opposed to property-oriented, as in the case of SIDEARM. One only needs to verify that the KIS adheres to some low-level properties. For example, the KIS must be shown to totally clear the CPU registers during a context switch to ensure the removal of all residue information.

3. Kernel Extension (KE)

KE's implement

application-specific and machine-specific portions of the security policy. For example, labelling output for peripherals such as printers and terminals is highly dependent on the type of device. Yet proper labelling is absolutely imperative to MLS (see paragraph I.2.). All of the expensive controls within the system are for naught if someone can determine or alter the label of data output. If that occurred, a process could then wantonly downgrade information to unclassified by improper output labelling. In this capacity, the KE implements the nongeneric portion of the TCB so that the SIDEARM can be architecture-independent.

KE's are also used to implement application-specific security policy. For example, a computer may have a MLS DBMS. A DBMS requires extra controls over and above the controls imposed by the operating system to restrict inference and aggregation. All application-specific extensions to the security model cannot be included in the reference monitor simply because the reference monitor would become too large. Further, we can not predict all the possible policy extension required for applications as yet un-built.

The KE's provide application-specific, security-related services as needed. The KE's, therefore, interface to potentially hostile applications and implement their special services at the control of SIDEARM via calls through the KIS. Since KE's capture machine-specific and application-specific portions of the TCB, they will be minimally portable between different types of LOCK computer systems.

These KE's may have some privileges not associated with normal programs. For example, a downgrader KE has the privilege to violate the *-property. The KE's, however, are only given the privilege required to do their task and are verified not to abuse that privilege. In short, KE's are the flexible part of the TCB, under strict control by the hardware reference monitor - the SIDEARM.

4. UNIX

The operating system in LOCK is considered hostile code. This means that the operating system will not have to be reverified and recertified after updates and changes as is the case with traditional software kernel approaches to computer security.

Does this mean you can take an existing operating system on a machine, retrofit it with a SIDEARM, and simply run the operating system unaltered? No! Operating systems typically perform security-related functions rooted in resource (CPU time, memory) management. These parts of the operating system will now have to be performed by the SIDEARM. Therefore, some of the operating system internals will have to be removed and replaced by calls into the KIS which, in turn, calls the SIDEARM.

UNIX was chosen to demonstrate the principles of LOCK because it is relatively small and because it is a

popular operating system. Since the operating system is treated as a hostile application and the interface to the KIS should be fairly stable, the implementation of a different operating system on the LOCK base should not prove difficult. Furthermore, once the KIS is developed for a different computer system, the modified UNIX should easily port to the new computer system.

What about applications software portability? Since LOCK is intended to be an AI certifiable computer, the question of necessary changes to the UNIX System V interface definition arises. It is pretty clear that it will not be possible to leave the interface completely unaffected and have AI security. We do not yet know what the full impact to the operating system and applications will be. It is our goal to minimize the impact and to localize it so that any changes necessary in porting an application from an unmodified UNIX to our Secure UNIX will be reasonably small and perhaps automatable.

H. Verification

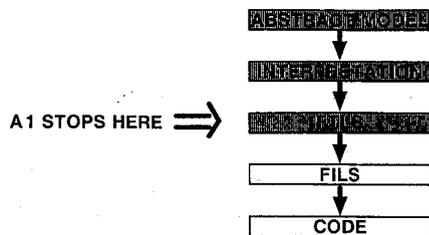
LOCK is using the Gypsy verification environment [GOOD78] to prove its security properties. The general approach that has been taken is to prove the system top-down in conjunction with the system design.[BOEB85a, BOEB85d] As the design is refined from the preliminary, highly abstract level to the more precise, detailed level, the verification proofs are proceeding at the same pace (or even slightly ahead of the design) and are used to feedback critical design issues. These issues arise in places where the verification team is having difficulty proving security. Feedback will show where the team can simplify the design, making the proofs easier and conceptually cleaner.

The use of type enforcement (see paragraph I.) makes the proof of the system security much easier. The type enforcement mechanism allows us to prove the unbyassability and tamper resistance of process modules. By proving this once, we can carry the lemmas over to other sections. This allows us to focus on the correctness of the next piece we must prove. It also permits a much greater assurance in trusted processes, as their privileges may be precisely given, and thus, restricted.

There are three established levels of proof. The abstract model, which is very general; the interpretation level, somewhat more detailed; and the formal top level specification (FTLS), the most detailed yet (see Figure h).[HONE86] The FTLS is complete except for some modifications, and the addition of most of the kernel extensions are still to be done. Along with these formal proofs is a Descriptive Top Level Specification -- a document that explains the security-relevant features in English narrative statements.

1. Formal Implementation Level Specification (FILS)

TOP-DOWN SPECIFICATION AND PROOF



- REPROOF OF SPECIFICATION AT EACH LEVEL
- MAPPING BETWEEN EACH LEVEL
- CRITERIA REQUIREMENTS STOP AT THE FTLs LEVEL OF DETAIL

Figure h

Although the Criteria calls for the FTLs to be the lowest level of proof, the LOCK team will probe to a deeper level. This next level we call the Formal Implementation Level Specification (FILS). Whereas the FTLs presents the TCB-user interface without the details of how that interface is implemented, the FILS delves into the implementation detail or the internal workings of the TCB. The FILS will essentially be a very detailed specification for the TCB and related code. This level of detail will also facilitate the required specification-to-code mappings.

2. Noninterference

The proofs of security are based on Goguen and Meseguer's notion of noninterference.[GOGU84] The noninterference model is an information-based model as opposed to an access control based model.[BELL75] Simply stated, noninterference requires proof that a subject cannot interfere with anything a lower-level subject can view in the system. This prevents any flow of information (assuming the security system is modeled correctly and completely) from a high to a low level and thus closes many covert channels that would exist in access control based models.[HAIG86]

The proof of noninterference is based on a recently-developed unwinding theorem. The verifier must prove that the low-level subject's output from his instructions are not affected if the corresponding instructions from a high-level subject are deleted from the instruction stream being input to the reference monitor. Since strict noninterference would severely inhibit system operability, the LOCK verifiers will be using a modified form of noninterference called conditional noninterference. This states that the low-level subject's output is not affected except in specific instances. These instances will be the only place where covert channels may occur, assuming perfectly faithful implementation of design.

The LOCK effort will also produce a covert channel analysis tool using the shared resource matrix methodology in conjunction with a Gypsy flow analyzer. This tool will be used on those sections where the noninterference proofs cannot be

proved easily and will identify the covert channel involved. While it will fail where a covert channel exists, noninterference does not identify the channel - hence the need for this tool.

Noninterference is applied both in the multilevel security sense and in the multidomain (see paragraph I.1.) security sense. A noninterference proof is sufficient to assure proper access control and the absence of covert channels between classification levels and between different domains. This allows the type enforcement to be just as "tight" security-wise, as MAC.

An auxiliary step in the verification process will be for the LOCK verifiers to couch selected Gypsy proofs in mathematical journal level language and submit them to a social review process.[BOEB85d] This allows the proofs to be looked at by someone not necessarily familiar with the intricacies and peculiarities of automated theorem provers, as well as forcing us to give a less abstruse or esoteric proof.

I. Type Enforcement

Type Enforcement is LOCK's way of providing mandatory, configurable integrity. Both DAC and MAC may be thought of as mechanisms that restrict access to information. Type enforcement is just another restriction placed upon the results of the first two. Access rights are whatever passes the three "filters." Type enforcement relies on the use of levels and labels on subjects and objects and has rules to permit access. This information is encoded within a matrix similar to the normal MAC matrix.[BOEB85b]

1. Domain Definition

Table/Domain Transition Table DDT/DTT

Type enforcement relies on two data structures: the DDT and the DTT. The DDT is a matrix with "types" on one axis and "domains" on the other. The intersection is the set of privileges (possibly null) a subject within a particular domain has to an object of a particular type. The DTT is a matrix with "subject" labeled on both axes; "subject," in LOCK terminology, is a user-domain pair. The intersection is a simple "yes" or "no," indicating whether a particular user in a particular domain may transition to a different domain.

2. Assured Pipelines

The DDT/DTT may be configured such that an "assured pipeline" is created. This is a control structure wherein an object is input to the lead control process, undergoes some intermediate processing, and is finally output in a "refined" form. The LOCK has the ability to insure that no unauthorized process may contaminate the object as it moves through this pipeline.

An example of such a construct would be a system where all the output from a printer is labeled with the proper classification label (as required by

the Criteria). One desires a system where it is easy to prove that text to be printed is labeled before being printed. Also, one has to prove that no information may be printed without being labeled, and that no process can change the label on a labeled file before it is printed.

The setup for the DDT is shown in Figure i. Any normal user may read and write to something called "raw text." When he wishes to print this, the labeler may read from this object (type: raw text) and write to an intermediate object (type: labeled-for-printer text). The printer may read only from objects of type labeled-for-printer text. No other rights for the two types are given to any other domain. This forces objects to be labeled before being printed and assures us that only the labeler has touched the label. A graphical representation appears in Figure j.

DDT	Labeled-for-Printer Text			Printer
	Raw Text			
USER DOMAIN	r/w			
LABELER DOMAIN	r		w	
PRINT SPOOLER DOMAIN			r	w

THE IMPLEMENTATION OF THE ASSURED PIPELINE MERELY REQUIRES AN APPROPRIATE CONFIGURATION OF THE DOMAIN DEFINITION TABLE.

Figure i

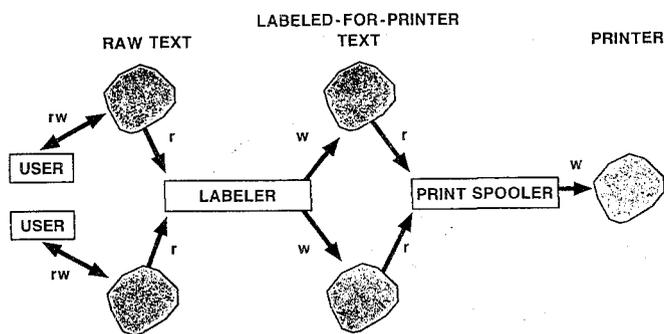
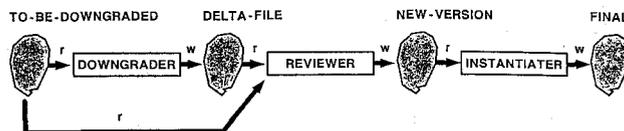


Figure j

Another pipeline that illustrates the utility of the assured pipeline is the downgrader example (see Figures k & l). This can be viewed as a triple turn-key operation. If the downgrader, the reviewer, and the instantiator are all different users, it requires that all three take some positive, specified action to allow the object to be downgraded.

DOWNGRADER



- DOWNGRADER -- CREATES FIRST DRAFT OF CHANGES REQUIRED FOR DOWNGRADING
- REVIEWER -- COMPOSES DRAFT WITH ORIGINAL AND OPTIONALLY CREATES A REVISION
- INSTANTIATER -- REVIEWS DRAFT 2 AND PERFORMS THE ACTUAL DOWNGRADE

Figure k

DDT	DELTA FILE		FINAL	
	TO-BE-DOWNGRADED	NEW-VERSION		
DDT				
DOWNGRADER DOMAIN	r	w		
REVIEWER DOMAIN	r	r	w	
INSTANTIATER DOMAIN			r	w

Figure l

3. Verification

Encapsulation

Type enforcement and the ability to construct an assured pipeline leads to an important encapsulation for verification purposes. Once the type enforcement is proved to work correctly, other proofs may use type enforcement to satisfy some security properties. Type enforcement may be used to satisfy the nonbypassability requirement as well as the tamper-proof requirement. Since subjects are restricted from accessing certain objects by the DDT and this concept can be extended to provide the pipeline effect, it can be shown which processes will have what access to what information in a very precise manner. It is possible to have a single program occupy a domain. This level of granularity is unmatched by MAC and is unmatched by DAC in assurance.[BOEB85c] A proof of the type enforcement mechanism, then, leads to a simplified proof of two of the three properties for modules in a reference monitor.

J. LOCK Applications

As mentioned earlier, LOCK will be useful for different applications. In some cases, it will be necessary to provide more than the generic portion, SIDEARM, to suit the needs of the user

completely. KE's will be the mechanism employed to extend the generic portion to handle the specific applications. This allows the computer system designer to incorporate only the functionality that is required by the users. For example, if the system will not be connected to a computer network in its life cycle, it would be unnecessary to incorporate secure networking functionality. The KE mechanism is a cost effective solution in addition to being an efficient means for implementing security functionality. Providing only the necessary security functionality eliminates the cost of additional bells and whistles not needed in particular implementation. Similarly, the addition of only the necessary KE's does not trigger exponential cost increases. Such is not the case for systems that incorporate mechanisms to handle all possible application-specific security. Another cost-reduction factor is that the KE's are verified and certified to the extent that they perform their job correctly and are dynamic entities that can be added without causing the entire system to undergo reverification and reevaluation.

The type enforcement mechanism allows the encapsulation of subsystems and the implementation of assured pipelines. This makes the development of multilevel applications much easier in addition to providing a higher degree of assurance. Currently, three ongoing efforts demonstrate LOCK's applicability and are described below.

1. BLACKER

The BLACKER project provides multilevel security for packet-switched computer communication networks through the use of a front-end interface between the host computer and the network. BLACKER achieves network security by maintaining secure electronic key distribution as well as end-to-end encryption. While BLACKER provides security between computers, LOCK provides security within the computer. The KE mechanism within LOCK can be used to emulate the BLACKER functionality to allow secure computers to communicate securely. The specifics required by BLACKER can be incorporated into a KE designed to provide the necessary functionality for communication on a BLACKER network if desired.

2. Secure Data Network System (SDNS)

The SDNS project is a strategy for securing communications over public data networks through end-to-end encryption. The manner in which this is done is vastly different than that employed by the BLACKER project. SDNS utilizes a different form of key management and distribution than the one implemented by BLACKER.

A major contribution of the SDNS project is a user-friendly, user-transparent key management technology. This key management strategy does not employ any access control functionality. SIDEARM, the foundation of LOCK, provides a very rich

access control facility capable of handling this task without the addition of an access control module. An SDNS KE would have to be specified, verified, and implemented to allow communication on an SDNS network. Transition between the two forms of secure networks discussed above becomes a matter of using the BLACKER KE set or the SDNS KE set.

Eventually, the technology produced in this effort will undergo a Commercial COMSEC Endorsement Program (CCEP) security product development [BARK86] to provide the opportunity for industry and government to develop a security architecture compatible with the Open Systems Interconnection network model [ZIMM80] developed by the International Standards Organization.

3. Secure Distributed Data Views (SDDV)

The Honeywell SDDV project is a multilevel secure relational database management system (MLS/DBMS) designed to run on the LOCK TCB. Secure database systems are application software that manage large amounts of information at different classification levels. The MLS/DBMS security policy is developed as an extension to the LOCK base policy and will be implemented via the KE mechanism. SDDV will demonstrate the advantages of LOCK for developing MLS applications as it is the first application built upon the LOCK foundation. The type enforcement mechanism isolates the components of the DBMS, making the operations fairly simple extensions. SDDV takes advantage of the assured pipelines and the modular structure that LOCK provides to implement the kernel extensions in a secure, flexible, and functional manner (see Figure m). SDDV is contracted by the Air Force Rome Air Development Center (RADC) to the Honeywell Secure Computing Technology Center and Stanford Research Institute. Success of the Honeywell effort depends upon the LOCK technology being fully developed.

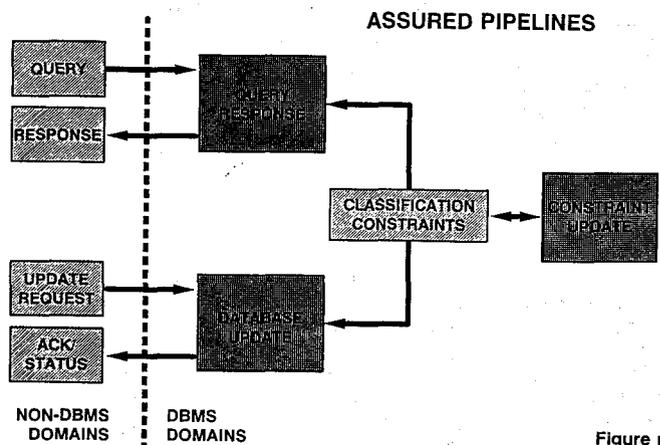


Figure m

III. PLANS

The LOCK program strives to lay down a foundation in computer security that all others may stand on. We are concentrating substantial time and money to develop a generic, reusable module that will meet a significant percentage of the Criteria's requirements. Implementors using this module would then only need to fulfill the remaining requirements in their system.

By the end of 1988, an Interface Control Document (ICD) for the SIDEARM will be published to allow major DoD and civil agency programs as well as the private sector to incorporate the requirement of connection to SIDEARM in their future systems. It will also allow computer vendors to initiate retrofits of current computer systems if it proves cost beneficial.

The National Computer Security Center's (NCSC's) sponsorship of LOCK is to provide risk subsidization for the research and development required for the first design and implementation of the secure modular technology. The intent is that private industry will pick up the gauntlet and fund development for retrofits and future-fits on their own systems to enhance the security and, thus, marketability of their computers. The standard NCSC and Commercial COMSEC endorsement processes will be followed in certification of such products.

Development on both sides of the ICD boundary is envisioned. Incorporation of the generic modules into a given system involves development on the host computer side of the boundary. Application of VLSI and VHSIC technologies to the implementation of the reference monitor module involves development on the SIDEARM side of the boundary. There are vendors who are capable and interested in both types of development work.

Even before first prototypes have been designed and implemented, secure applications targeted for LOCK computers are being developed for securing databases. The RADC-sponsored SDDV project is designed to tie-in to the LOCK module base. The estimated cost savings of using an existing TCB versus a ground-up design is significant.

Even after there are hundreds of different computer systems that have successfully integrated the embedded SIDEARM module and they are placed on the NCSC Evaluated Products List (EPL), the work is not done. The LOCK technology provides the mechanisms required to implement integrated secure applications on top of the TCB provided by the modules. LOCK will form the foundation for computer security for many requirements for many years to come.

BIBLIOGRAPHY

- BARK86 Barker, William Curtis. "An Industry Perspective of the CCEP." AIAA/ASIS/DODCI Second Aerospace Computer Security Conference, Dec. 2-4, 1986.
- BELL75 Bell, David E., and L. J. LaPadula. "Secure Computer Systems: Unified Exposition and Multics Interpretations." Technical Report MTR-2997, Mitre Corp., July 1975
- BOEB85a Boebert, W. E., et al. "Secure Ada Target: Issues, System Design, and Verification." Proceedings of the 1985 Symposium on Security and Privacy. IEEE, 1985.
- BOEB85b Boebert, W. E., and R. Y. Kain. "A Practical Alternative to Hierarchical Integrity Policies." Proceedings of the 8th National Computer Security Conference. Oct. 1985.
- BOEB85c Boebert, W. E., and C. T. Ferguson. "A Partial Solution to the Discretionary Trojan Horse Problem." Proceedings of the 8th National Computer Security Conference. Oct. 1985.
- BOEB85d Boebert, W. E., et al. "Secure Computing: The Secure Ada Target Approach." Scientific Honeyweller. June 1985.
- FERG86 Ferguson, Chuck, and C. B. Murphy. "A Proposed Policy for Dynamic Security Lattice Management." Proceedings of the 9th National Computer Security Conference. Sep 1986.
- FORD81 Ford Aerospace and Communications Corporation. "Provably Secure Operating System (PSOS) Final Report." Contract MDA 904-80-C-0470. June 1981.
- GOGU84 Goguen, Joseph, and Jose Meseguer. "Unwinding and Inference Control." Proceedings of the 1984 Symposium on Security and Privacy. IEEE, 1984.
- GOLD84 Gold, B. D., et al. "KVM/370 in Retrospect." Proceedings of the 1984 Symposium on Security and Privacy. IEEE, 1984.
- GOOD78 Good, D. I., et al. "Report on the Language Gypsy, Version 2.0." Technical Report ICSCA-CMP_10. Institute for Computing Science, Univ. of Texas at Austin, Sep. 1978.

HAIG86 Haigh, Tom, and W. D. Young. "Extending the Non-interference Version of MLS for SAT." Proceedings of the 1986 Symposium on Security and Privacy.

HONE83 Honeywell. SCTC A-Specification, Contract MDA-904-82-C-0444. April 1983.

HONE86 Honeywell. SCTC B-Specification, Contract MDA-904-84-C-6011. March 1986.

KIBA86 Kibalo, Thomas, and W. E. Boebert. "Using Embedded COMSEC: an Integrator's Viewpoint." Proceedings of the 2nd Annual Symposium on Physical/Electronic Security. August 1986.

MCHU85 John McHugh, "An Emacs-Based Downgrader for the SAT," Proceedings of the 8th National Computer Security Conference, Oct. 1985.

MYER80 Myers, Phillip A. "Subversion: The Neglected Aspect of Computer Security." Thesis, Naval Postgraduate School. June 1980.

NEUM77 Neumann, P. G., et al. "A Provably Secure Operating System: The System, its Applications, and Proofs." Final Report, Project 4332. SRI, Feb. 1977.

SAYD86 Saydjari, O. Sami, and Tim W. Kremann. "A Standard Notation in Computer Security Models." Proceedings of the 9th National Computer Security Conference. Sep. 1986.

TCSE85 Department of Defense Trusted Computer System Evaluation Criteria. DOD 5200.28.STD. Dec. 1985.

YOUN85 William D. Young et al., "Proving a Computer System Secure," Scientific Honeyweller, Vol. 6, No. 2, July 1985.

YOUN86 William D. Young et al., "A Verified Labeler for the Secure Ada Target," Proceedings of the 9th National Computer Security Conference, Sep. 1986.

ZIMM80 Zimmermann, H. "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection." Transactions on Communications, IEEE, April 1980, Vol. COM-28, pp. 425-432.

UNIX and B2: Are They Compatible?

W. Olin Sibert
Oxford Systems, Inc.
Arlington, Massachusetts
617 646 8619

Holly M. Traxler
Grant M. Wagner
National Computer Security Center
Fort George G. Meade, Maryland

Dr. Deborah D. Downs
Kenneth B. Elliott, III
The Aerospace Corporation
Los Angeles, California
213 336 4365

Jeffrey J. Glass
The MITRE Corporation
Bedford, Massachusetts

ABSTRACT

As an emerging operating system standard, UNIX is being used more and more as a basis for building secure systems. In late 1986, the National Computer Security Center (NCSC) studied a prototype secure system derived from AT&T's System V, Release 2 version of UNIX. The study assessed that system's compatibility with the B2 assurance requirements defined in the Trusted Computer System Evaluation Criteria (TCSEC). The study also gave increased insight into the meaning of and relationships among those requirements. This paper presents the results of the study and some advice for builders of systems intended to meet the B2 requirements.

The views and opinions expressed in this paper are solely those of the authors and do not necessarily reflect official National Computer Security Center positions. In particular, this paper should not be considered as an official Criterion Interpretation for any TCSEC requirements.

INTRODUCTION

Last year, the NCSC performed an extensive study of the implementation of a prototype secure system based on AT&T's System V, Release 2 version of UNIX. The study was performed by a five-person team, including experts in UNIX, operating system design, and security evaluation, during eight weeks in the fall of 1986. The purpose of this study was to evaluate the suitability of the generic UNIX system architecture as a base for building secure systems. Although the study was primarily concerned with a single example system, the team attempted wherever possible to generalize the results to other versions of UNIX.

This paper discusses why the study was conducted, why the B2 level of the TCSEC [DoD85] was chosen and what requirements of the TCSEC were considered. It then presents a definition of modularity that was used in the analysis of UNIX and reports the results of applying that definition, including several specific examples common to most UNIX systems. The paper concludes with recommendations on how to approach development of B2-class systems.

Outline of the Study

The NCSC is evaluating several UNIX-based or UNIX-like systems at various TCSEC levels and expects to be evaluating more in the future. This study was undertaken in the interest of ensuring that similar systems be evaluated consistently. The NCSC formed the study team to evaluate the internals of one of these candidate systems in depth. The purpose of the study was not to examine the particular system's security features for sufficient implementation, but rather to evaluate how well the candidate system and generic UNIX systems meet the architecture requirement and provide assurance of secure operation.

This work was supported in part by the National Computer Security Center under Task Order Number MDA903 84 C 0031:T-J5-XXX issued to the Institute for Defense Analysis.

UNIX is a registered trademark of AT&T.

While the study focused on a specific system, the results do appear applicable to most security-enhanced UNIX-based systems, and, in a larger sense, are applicable to all systems targeted for the B2 level or higher. The results specific to UNIX are applicable to any system built by modifying a standard release of UNIX to add security features without a complete internal restructuring both within the kernel and without.

The study team concluded that it is possible to build a B2, B3, or A1 system with an interface very much like that of UNIX. However, it also concluded that major problems exist with today's common UNIX implementations. The study also provided valuable insights into the meaning of the B2 architecture and other assurance requirements, which are applicable for any candidate B2 (or above) system, not just UNIX-based systems.

Why the B2 Requirements

The choice of B2 as a target level for the analysis was not arbitrary. Building a B1 system is, in principle, straightforward, since B1 primarily requires features, not assurances. The intent of the B1 rating is to establish a target that can be reached by modifications to many existing systems, while still providing mandatory access control features.

A B3 level analysis was not considered because of the stringent requirements for system structure and minimal TCB complexity. It seemed unlikely that a UNIX-based system would even approach this level of layering and minimization except through complete reimplementations.

It seemed possible, however, that the B2 level *could* be achieved in a UNIX-based system without wholesale reimplementations. Although B2 is intended as an evaluation class for systems designed from the beginning with security assurances in mind, UNIX is among the few existing systems that could possibly be adapted to include those assurances without a complete reimplementations of its Trusted Computing Base (TCB). UNIX systems have already served as an implementation base or interface standard for many computer security research projects (UCLA Data Secure UNIX, KSOS, SCOMP UNIX emulator).

The study team's primary focus was on System Architecture, and modularity in particular, where the System Architecture requirement calls for "well-defined, largely independent modules." The modularity of a system is fundamental, and cannot be improved simply by adding features. Modularity is also relatively independent of the hardware base and even, for the most part, of the strategy for adding security features. The modularity analysis represented the bulk of the team's effort, and its results are presented in the next section ("UNIX MODULARITY").

All of the B2 functional requirements, and to a large degree even the other assurance requirements (besides System Architecture), are concerned with aspects of the system which are likely to differ substantially in different implementations. Since the team's charter was limited to studying aspects of the system which could be considered "generic", and common to most UNIX-based secure systems, no consideration was given to specific security features of the examined system or to implementation-dependent assurances. Only those aspects which are largely independent of any one implementation were examined; this included several assurance areas in addition to architecture, and these are discussed in the "Additional Assurance Areas" section below. The remaining assurance areas (such as covert channel analysis, formal model, testing, and configuration management) were found to be too specific to a particular system's design and implementation for a generic analysis to have much meaning.

UNIX MODULARITY

Most of the study team's analysis reviewed modularity and independence of the components of the UNIX kernel, along with some of the trusted processes (e.g., *mkdir*, printer spooler). A definition of modularity had to be chosen and a determination made on how it could be applied to the UNIX system. That definition was used to evaluate TCB components. The study team conducted a detailed analysis of nearly one third of the UNIX TCB, and examined most of the rest.

The first part of the assessment required choosing a definition for "modularity" and applying it to the system. Modularity can be present at many levels. At a very low level, the instruction opcodes of a processor could be considered "modules." At a very high level of abstraction, the entire kernel could be considered a "module" whose interface and parameters are described by the DTLs. To understand a system, an appropriate level of abstraction must be chosen, and this proved rather difficult. The team wanted to look at an abstract level of "major subsystems" in the kernel (such as file I/O, directory management, memory management [Bach86]) but found this to be impractical. Although the existence of such subsystems can be argued from a functional standpoint, no such clean boundaries exist within the kernel. Instead, many individual functions directly manipulate whatever data structures they must to "get the job done." In the absence of more abstract modules, the team was forced to use a lower level of abstraction. This less abstract view required treating individual C-language functions (and occasional assembler-coded functions) as modules and evaluating them against their definition. This choice was made largely because the functions were readily identifiable in the source code.

The study team also started with a definition of "modularity" that claimed modularity is independent of packaging. In theory, this is reasonable, but the originally poor packaging of the system and of the code added to support security features made this claim unsupported. The packaging problems make the system more difficult to maintain, thus resulting in potentially lower security. The packaging also made it very difficult to identify modules at a higher level of abstraction than a single function, since it was almost impossible to find all of the components of the more abstract module.

Ultimately, the team concluded that modularity must not only be present, but must also be readily apparent to a skilled observer. Hidden modularity is of no value; for instance, an uncommented disassembly listing of a

system which is fully modular in its high-level language form could not be considered modular even though the two forms are completely equivalent from the machine's point of view. Modularity cannot be just an artifact; it must be an expressed and evident intention. It must be present in the implementation, supported by the design documentation, and maintained through the configuration management system.

Definition of Modularity

The team's original working definition of "module" was:

"... a conceptual building block that corresponds to the work assignment of a programmer or programming team... a group of closely related programs."

As described above, however, this definition proved very difficult to use. The "work assignment" model is too vague. Because of the extensive use of global variables in the kernel, very large parts of the kernel become closely related. The "modules" that result from this definition were too large and too diverse functionally to be considered modular.

Therefore, in the basic analysis of the system, the team used a module definition similar to Stevens, Myers and Constantine [Stevens74], in which a module is:

"a set of one or more contiguous program statements having a name by which other parts of the system can invoke it and preferably having its own distinct set of variable names."

The basic assumption of the analysis was that if all the modules in an operating system met the following criteria, the system could be considered fully modular. A module:

- performs exactly one well-defined function;
- has well-defined parameters, interface and environment;
- interacts with other modules only in well-defined ways; and
- is called upon to perform its function whenever that function is required.

The first criterion means that a module should not combine multiple functions, particularly if they are unrelated or are also performed in other modules, and also that the results of a module should be predictable, based solely on the values of its input parameters. In [Stevens74], under the heading "Functional binding", is an interesting first cut methodology which can be used as an aid in determining if a module meets the first criterion. It is based on evaluating a one sentence description of the module. Although the study team did not use this technique, it appears very useful, as an aid to building or restructuring a secure system.

The second criterion means that the interface to a module should clearly reflect its implementation. There should be no hidden dependencies on or assumptions about other parts of the system (e.g., arrangement of data in memory, internal operation of unrelated modules, details of hardware, etc.). The module name and its parameters should give a fair guess about its function. As Britton and Parnas wrote in [Britton81],

"A software engineer should be able to understand the responsibility of a module without understanding the module's internal design."

The third criterion is related to the second in that parameters passed to and returned from a module should be clearly identified and have well-defined consistent meanings. Parameters, formal and informal (e.g. global variables or environment), are an important part of the connections between modules. From [Stevens74],

"Minimizing connections between modules also minimizes the paths along which changes and errors can propagate into other parts of the system, thus eliminating disastrous 'ripple' effects where changes in one part cause errors in another, necessitating additional changes elsewhere, giving rise to new errors, etc."

Finally, B2 is a popular target. The NCSC currently is experiencing considerable demand for B2-level developmental evaluations of many types of systems. Because the B2 assurance requirements in the TCSEC are vague and difficult to interpret, one of the goals of the study was to define them more clearly in support of those evaluations. Honeywell's Multics, as the prototypical B2 system, is considered to have met the B2 requirements, but it is just a single sample point and cannot provide all the needed guidance. As hoped, the UNIX study has provided additional guidance on how to interpret the B2 requirements and apply them to systems under evaluation.

REVIEW OF THE B2 REQUIREMENTS

Having chosen the B2 requirements as the target level for the analysis, the study team's first task was to decide how to assess compliance with those requirements. This was done in two stages: first understanding the important distinctions between B1 and B2 then identifying the essence of the B2 requirements.

The Difference Between B1 and B2

From reading the TCSEC, it appears that the chief distinction between B1 and B2 is one of assurances, and of the comprehensiveness of those assurances. The step from B1 to B2 is regarded as the single most difficult transition in the Criteria, principally because of these assurance requirements.

The intent of these assurances is to eliminate vulnerabilities to attack, real or potential, that do exist or that *might come into existence* during the entire life-cycle of the system — or, at least, minimize the likelihood of such vulnerabilities. In the early 1970's, in [Anderson72] and other work, it was observed that the mere appearance of correct functioning, or even of correct implementation, was not sufficient to ensure that a system would operate securely under attack; nor was that appearance sufficient to ensure that a once-secure system would remain secure after additional development. From this work, the overall goal of an inherently secure, understandable, and maintainable system emerged: the Reference Monitor.

When examined in this light, the Reference Monitor was seen as the *fundamental* assurance of a B2 system. In effect, all the other assurance requirements exist to define, support, maintain, and protect the reference monitor: specification, testing, maintenance, and architecture.

Furthermore, although not explicitly stated in the TCSEC, it became clear that not only must the system meet the assurance requirements, but that it must be *evident* that it does so.

Critical B2 Requirements

Five of the B2 requirements appear to constitute the essence of the B2 assurances. Although several requirements differ between B1 and B2, and others are differently interpreted because of the B2 assurances, these differences are either direct and obvious consequences of the B2 assurances or requirements for features that have no direct relationship to those assurances. The primary functional difference, covered by the System Architecture requirement, is the need to apply the TCB's security policy to all subjects and objects, rather than just a chosen subset.

The five critical B2 requirements are:

- System Architecture
- Design Documentation
- Design Specification and Verification
- Covert Channel Analysis
- Configuration Management

These five requirements are (at best) vaguely written and all closely related. This makes them difficult to consider individually. Therefore, for this study, they were instead considered as a whole and their contents reorganized into twelve areas; some of the areas also incorporate small extracts from other requirements. Each of these twelve areas represents a well-defined goal to follow when planning the development of a secure system, unlike the original five requirements. The appendix at the end of this paper contains the complete set of extracts for each of the areas, and a brief summary of each.

The twelve assurance areas identified by the study team are:

- Reference Monitor Requirements
- TCB Functional Requirements
- TCB Isolation Requirements
- Process Isolation Requirements
- Modularity Requirements
- Least Privilege Requirements
- Hardware Requirements
- Descriptive Top-Level Specification Requirements
- Configuration Management Requirements
- Covert Channel Requirements
- Formal Model Requirements
- Testing Requirements

SCOPE OF STUDY TEAM'S ANALYSIS

The object of the team's analysis was the Trusted Computing Base (TCB) software of a UNIX implementation. The generic UNIX TCB consists of two major software components, the kernel and the trusted processes. An actual product evaluation of a real UNIX system would also consider the hardware and firmware components of the TCB, but since the study was concerned only with the generic UNIX software architecture, those components were not considered. The UNIX TCB software is written primarily in the C programming language, although there is some amount of implementation-dependent assembler code in most UNIX systems.

The first software component is the kernel, which runs in the hardware's privileged state. Its services are requested by "system calls", which switch the process to run in the privileged state and transfer to the kernel code. Its data structures are shared by all processes, but accessible only when the process is running in the kernel. The kernel implements most of the basic operating system functions that control sharing of and access to resources.

The second software component is the set of trusted processes. The trusted processes are simply all those UNIX processes which run with a distinguished (privileged) user ID. One such user ID is zero, the *root* user ID, or the "super user", which can execute all privileged system calls and is not subject to access control. Other distinguished user IDs are used to grant access to certain shared TCB files and directories, such as the printer spooler's directory (though in standard UNIX, most such access is granted only to *root* and not more tightly controlled). All other processes (belonging to unprivileged users) are outside the TCB, although an unprivileged process may dynamically invoke a privileged (trusted) process to request some service from the TCB.

When considering the actual UNIX implementation, the study team quickly realized that the primary question was *not* whether the system included a reference monitor, whether it isolated the TCB, whether it was composed of well-defined modules, and so forth, but rather, *how* to make that determination. It was simply not *evident* that any of these requirements were met, and difficult to see how one could make that determination, or ensure that the requirements would still be met after maintenance in the future. Although a UNIX expert might argue that all these architecture requirements *are* met, had one but the wit to see it, this merely reinforces the conclusion that the system's architecture is not manifest in its implementation, but instead exists primarily in the minds of its designers.

Good modularity minimizes coupling. Coupling [Stevens74] is:

“a measure of the strength of association established by a connection from one module to another . . . Coupling increases with increasing complexity or obscurity of interface.”

Extensive use of global variables causes every module sharing them to be coupled to every other such module “without regard to their functional relationship or its absence.” [Stevens74] Belady and Lehman [Belady71] observed that:

“a well-structured system, one in which communication is via passed parameters through defined interfaces, is likely to be more growable and require less effort to maintain than one making extensive use of global or shared variables.”

Britton and Parnas [Britton81] also expressed their views of global variables. This definition is much closer to the original working definition, but is also at a lower level of abstraction than simply considering the entire TCB as a single module.

“Every data structure is private to one module; it may be directly accessed by one or more programs within the module but not by programs outside the module.”

Of course, an operating system may require some use of global variables. But as [Stevens74] observes,

“it is possible to minimize the disadvantages of common environments by limiting access to the smallest possible subset of modules.”

The fourth criterion means that the function performed by a module should always be performed by calling that module, rather than being implemented in multiple locations. Note that this does not preclude inline macro expansions of code inserted from include files; rather, the goal is simply to ensure that the actual programmer-created definition of a function appears in only one place.

The desire that multiple uses of a function share one definition should not be taken to mean that a function required only once ought not to be defined independently. Rather, wherever possible, independent functions should be implemented as separate modules, even if they are only used once. This criterion, along with the use of consistent programming style among developers, should result in a similar control structure and call sequence in modules performing similar functions.

UNIX System Modularity

The study team’s examination of the UNIX TCB covered approximately 60 percent of the kernel and 35 percent of the trusted (privileged) processes. For every function examined in the TCB, the team produced an analysis report describing the function’s apparent contract (as best as could be determined from reading the code), its parameters, its use of global variables, its calls to other functions, and its conformance with the definition of modularity given above.

At higher levels of abstraction, the team looked for object or resource managers acting as the interface to TCB data structures wherever the packaging and structure supported such analysis. Although packages (C-language source files) also represent a readily identifiable group of code, the generic UNIX packaging of functions into source files is haphazard and uninformative (Berkeley UNIX has made some improvement in this area). The conclusion, however, was that while source file packages could form the basis for a more abstract view of kernel modularity, most of the basic UNIX TCB still would not exhibit a high-level modularity even if entirely reorganized and repackaged. The lack of clear divisions between major subsystems appears to be an inherent system characteristic, not merely an artifact of poor packaging. There were some notable exceptions to this observation, such as the file system’s buffer manager [Bach86].

After this detailed examination of the UNIX TCB, the study team concluded that generic UNIX does not meet the B2 requirements for modularity (in addition to problems with some other B2 requirements, discussed below). This conclusion was based on the following observations:

- The kernel includes pervasive misuse of global variables. It modifies supposedly constant values to take advantage of their side-effects, it shares global variables among wholly unrelated modules, it uses global variables in many cases where formal parameters should be used, and it uses global variables for temporary storage of purely local information. This is the single largest problem area, and the first one that should be corrected in any B2 UNIX implementation. A prime example of this was the global temporary values used by *namei* (the multi-purpose pathname resolution function). Global variables, as such, are much less of a problem for the trusted processes, since they primarily share data in files.
- The UNIX TCB contains numerous examples of duplicated functions—very similar, or in some cases identical—functions. These duplicated functions were sometimes syntactically identical, sometimes subtly different. These examples were often the result of using in-line operations rather than function calls to perform well-defined operations, such as searching for entries in the *mount* and *proc* tables. This was also a problem in the trusted processes, where a group of related trusted processes (such as components of the printer spooler) contained wholly duplicated code, rather than calls to library routines.
- The packaging of the kernel is very poor. Functions are scattered among different source files even though their purposes are closely related. This in itself does not prevent high-level modularity, but if not corrected would make the modularity invisible even if theoretically present. For example, functions for performing directory management are scattered among several different source files, as are the functions which perform access checks. Packaging of the trusted processes is much better, since each trusted process must have its own source file.
- Related to the above problem is the organization of external data structure definitions (“.h”) files. Practically every function in the kernel includes all the major “.h” files; determining whether a function actually references one of those data structures is therefore very difficult. This is an artifact of the C-language data definitions, but one which could be avoided by better structure and some help from the compilers. Here again, what data structures are actually referenced is not as important as ensuring that those patterns are readily apparent. Another aspect of this problem is that it is very difficult to determine (from name or usage) what modules “own” a particular structure element (or even the whole structure). This was much less of a problem for the trusted processes, since they share little (if any) data except by common file formats.
- Many TCB functions are complex and ill-defined. Rather than calling on another appropriate function to manipulate a data structure, the manipulations are done directly, with no assurance that they conform to the rules of the data structure’s managing functions. Other functions’ contracts are ill-defined, sometimes returning values or setting global values and other times not. Examples of this include the multi-purpose contract of *namei* and the complex series of operations performed by the *exec* system call or the *login* trusted process.

ADDITIONAL ASSURANCE REQUIREMENTS

Although UNIX was not specifically assessed in the assurance areas other than modularity (because of their implementation-dependent nature), the team concluded that UNIX-based systems were likely to require considerable work in other areas before approaching compliance with the B2 requirements. These are strictly *assurance* requirements, not functional

requirements for features such as auditing and mandatory access control. They represent additional work required beyond simply implementing the B2 security features. Work is needed in the following areas:

- **Reference Monitor** — The existing UNIX reference monitor is distributed among many programs, some in the kernel and many others outside. While a single “reference monitor” controlling all forms of access to all types of objects may be impractical, the UNIX “reference monitors” are far more distributed than necessary. Access checks are made in-line throughout the TCB rather than by calls to any common access policy routine.
- **Effective Use of Protection Hardware** — The base UNIX system is inherently a two-state system. The original implementation, plus years of portability, have left UNIX strongly mired in a hardware world with two protection states and an unsophisticated addressing architecture. Considerable work appears necessary to build a UNIX-based system that can take proper advantage of the more sophisticated hardware available in today’s systems.
- **Least Privilege** — Standard UNIX systems completely fail the least privilege requirement. Within the TCB, the only mechanism for restricting the privilege of TCB components is process isolation, and that only affects TCB components outside the kernel (the trusted processes). Within the kernel, all programs are equally privileged, and, since all trusted processes in a standard UNIX run as *root*, they are all also equally privileged. At the user and administrator interface level only vestigial forms of least privilege exist: an administrative user possesses all privileges, by virtue of running as *root*. To satisfy the B2 requirements, some form of least privilege is required for trusted programs, and a mechanism is required to separate administrator and operator roles.
- **Descriptive Top-Level Specification (DTLS)** — The UNIX DTLS takes the form of manual pages for system calls and administrative commands. Although the existing standard UNIX documentation is a good start, it is somewhat incomplete for system calls and seriously incomplete for administrative interfaces. For a secure UNIX system, new documentation must be provided to include new security-related system calls and administrator interfaces, but the existing documentation must be improved as well to give more complete functional descriptions, lists of effects and error returns, etc.

ADVICE FOR B2 SYSTEM BUILDERS

The study team’s main conclusion — and this is just restating what has been said many times before — is that building a B2 system is a hard job. It is certainly possible, but remember:

Bringing an existing system to the B2 level is likely to be at least as difficult as building a brand new system.

It is not clear whether it is possible to retrofit the B2 assurances into a generic UNIX system. Doing so appears to require major reorganization of code and data structures, if not outright reimplementing of many parts of the TCB. This is even more true for non-UNIX-based operating systems: UNIX does appear to be better structured internally than most of today’s existing systems. The UNIX system interface, being relatively simple, is also much better suited to a highly secure system than the complex interfaces of some other operating systems.

One of the goals of this study was to define the architecture requirement for a B2 system in a language that vendors and evaluators could understand. That goal was not achieved: no cookbook-like set of guidelines for building a B2 system is available. As with most software engineering topics, precise measures simply do not exist. As Supreme Court Justice Potter Stewart once said, “I may not be able to define obscenity, but I know it when I see it.” The same applies to B2 architecture.

This study did, however, produce some good analysis techniques for generating the necessary supporting evidence and for guiding system development in the right directions. The recommended techniques include:

- Eliminate global variables whenever possible. When they must be used, assign them precisely defined semantics. Enforce those semantics, perhaps by coding standards and code review guidelines, perhaps by automated source or cross-reference analysis, perhaps by using segmentation hardware to enforce the semantics at run-time, too. All of this adds greatly to internal assurance, and proper treatment of globals also seems to encourage other practices for good modularity.
- Use packaging to illustrate levels of abstraction, not to obscure them. Make certain that the system’s structure and modularity are evident, and consistent, throughout.
- Develop and follow, *throughout the system*, naming conventions that make the purpose of functions and variables readily apparent. Remember that a naming convention that isn’t universally followed is in many ways worse than none at all. Pay as careful attention to this aspect of the design as to any other.
- Use the hardware to its best advantage. If the system does not use a hardware protection feature, it is probably not as secure as it ought to be. These features are supposed to reduce the cost of security, not increase it.
- Write *everything* down. Document the coding practices, the packaging rules, the data structure design principles, the module hierarchy. Make each module’s contract explicit, each data structure’s purpose clear, and honor those contracts when making changes. One of the biggest problems with analyzing UNIX is that none of those hidden assumptions were documented. While the original implementor of a function may have known just what it was supposed to do and why, the programmers who modified it afterward usually lacked access to that knowledge — as did the study team.
- Treat the assurance requirements for B2 as a whole. As the study team learned at the very beginning, assessing compliance with just one of the assurance requirements is pointless, because the security of the system depends intimately on all of them. The five critical assurance requirements are all equally important, and slighting any one of them will lead to serious problems.

APPENDIX: SUMMARY OF B2 REQUIREMENTS

This section divides the B2 requirements into 12 logical groups, each of which consists of sentences or extracts from some of the B2 TCSEC [DoD85] requirements (some extracts may appear more than once, in different groups). Each group begins with the relevant sentence(s) quoted from the various requirement(s), and follows with a brief explanation of the grouping’s intent.

Each extract is identified with the requirement from which it comes, by one of the following abbreviations (in brackets) at the end of the extract. The first five requirements are the critical requirements for architectural assurance, and are quoted in their entirety among the 12 groupings. The other six requirements do not specifically require architectural assurances, but imply certain architectural characteristics.

SA	System Architecture
DD	Design Documentation
DS&V	Design Specification and Verification
CM	Configuration Management
CCA	Covert Channel Analysis

AUDIT	Audit
LABELS	Labels
MAC	Mandatory Access Control
TFM	Trusted Facility Manual
ST	Security Testing
TD	Test Documentation

The other sixteen B2 requirements have no specific bearing on architectural assurances, and are not considered in this analysis. Except for audit, labels, and mandatory access control (which differ at B2 in requiring comprehensive coverage of all TCB-provided objects), the remaining sixteen include all the feature requirements, the system integrity requirement, and the security features user's guide requirement. All but four of these other requirements are unchanged in wording from the equivalent at B1.

REFERENCE MONITOR REQUIREMENTS

"Documentation shall describe how the TCB implements the reference monitor concept and give an explanation why it is tamper resistant, cannot be bypassed, and is correctly implemented." [DD]

"The TCB modules that contain the reference validation mechanism shall be identified." [TFM]

"The TCB shall enforce a mandatory access control policy over all resources (i.e., subjects, storage objects, and I/O devices) that are directly or indirectly accessible to the TCB." [MAC]

"The following requirements shall hold for all accesses between all subjects external to the TCB and all objects directly or indirectly accessible by these subjects:..." [MAC]

"Sensitivity labels associated with each ADP system resource (e.g., subject, storage object, ROM) that is directly or indirectly accessible by subjects external to the TCB shall be maintained by the TCB." [LABELS]

These requirements address the implementation of the Reference Monitor [Anderson72] principle. The first two extracts quoted above address the reference monitor principles; the latter three address the completeness of its coverage. The Reference Validation Mechanism is an implementation (of a reference monitor) "that validates each reference to data or programs by any user (program) against a list of authorized types of reference for that user." The Reference Validation Mechanism must satisfy the following requirements:

- 1) The Reference Validation Mechanism must be tamper resistant
- 2) The Reference Validation Mechanism must always be invoked
- 3) The Reference Validation Mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured.

Although not explicitly stated, it is clear from these requirements that the Trusted Computing Base in a B2 system may contain more than just the reference monitor. It may also contain other components that are not directly involved with mediation of user access to data, but which nonetheless must function correctly for the TCB to satisfy the other requirements. To provide the necessary assurance, however, all components of the TCB must be guaranteed not to interfere with operation of the reference monitor proper, and this means that the entire TCB must be sufficiently well-structured and well-defined to be analyzed and tested.

There is no requirement for a single identifiable hardware or software component that is the reference monitor. Rather, the reference monitor is the collection of reference validation mechanisms used for different types of objects. This includes both hardware validation of direct access to memory and software validation of access to TCB-defined objects by invoking the TCB. Higher levels of abstraction in the reference monitor can and should be built to depend on lower levels. For instance, TCB calls to manipulate higher-level (software) objects should be able to rely

on the lower-level hardware mechanisms to validate accesses to user memory (containing parameters, for instance).

TCB FUNCTIONAL REQUIREMENTS

"Documentation shall be available that provides a description of the manufacturer's philosophy of protection and an explanation of how this philosophy is translated into the TCB." [DD]

This requirement addresses the high-level structure of the TCB and the TCB interface; in effect, how the mechanisms required by the "feature" requirements are collected into a TCB that implements them.

The "philosophy of protection" must map to the other Criteria requirements, but there are many possible mappings. It must cover both the security features and the mechanisms for TCB protection and isolation.

TCB ISOLATION REQUIREMENTS

"The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures)." [SA]

"... all elements of the TCB [shall be] identified." [SA]

This requires that the TCB be isolated in at least one domain inaccessible to users. It does not require precisely one TCB domain; rather, the isolation of TCB components into individual protection domains is a decision made to satisfy the requirements for structure and independence of TCB modules.

The TCB isolation may be provided by different mechanisms for different TCB domains. How this relates to use of hardware is discussed below (see Hardware Requirements). Since the TCB consists of all code with the potential to violate the system security policy (that is, both code that is intentionally granted the privilege and code that inherits it from the invoking environment), all such code must be isolated from tampering. If a variety of mechanisms is used to provide this isolation (for example, hardware privilege, process privileges, access to special files and/or devices, special user identity), the TCB boundary is much more difficult to analyze (or even describe).

PROCESS ISOLATION REQUIREMENTS

"The TCB shall maintain process isolation through the provision of distinct address spaces under its [the TCB's] control." [SA]

The term "address space" here refers not only to the addressable main storage accessible to a process, but also to the addressability of other TCB-provided resources (objects). This does not require that the system function without ever sharing objects between processes, but simply that the TCB's mediation and control mechanisms always come in to play for all objects; that is, that all sharing must be intentional.

MODULARITY REQUIREMENTS

"The TCB shall be internally structured into well-defined largely independent modules." [SA]

"The interfaces between the TCB modules shall be described." [DD]

"During development and maintenance of the TCB, a configuration management system shall be in place that maintains control of changes to... other design data, implementation documentation, source code..." [CM]

This is a complex topic, and is addressed only vaguely by the TCSEC requirements. The requirements are deliberately vague, to avoid dictating implementation technique, but the basic emphasis is one of good struc-

ture and program design. It is not the intent of the B2 requirements to demand that the entire TCB follow a uniform standard of perfection, but rather to ensure that the TCB is largely in compliance with the requirements, and that there are no major violations of modularity and independence.

The configuration management requirements for maintenance of documentation are particularly important to modularity. Design documentation must be kept up to date with the code, and therefore must be updated whenever the code is updated. To make this updating easier, external design documentation should provide a view of the code that focuses on the overall purpose of modules and the interactions between modules, rather than the details of internal structure. When arranged this way, external documentation should be supplemented by internal documentation (e.g., comments) in the modules themselves to cover internal details (though still at a higher level of abstraction than the code itself).

It is not sufficient simply to assert that "the code is the documentation."

LEAST PRIVILEGE REQUIREMENTS

"The TCB modules shall be designed such that the principle of least privilege is enforced." [SA]

"Documentation shall describe how the TCB is structured ... to enforce least privilege." [DD]

These requirements refer to the principle of least privilege within the TCB; that is, the means by which one is assured that a part of the TCB cannot exercise privileges beyond those required for its specific function. It is important that some specific mechanism be used to provide this assurance. It is not sufficient simply to assert that no part of the TCB uses its privileges inappropriately. It must also be possible to demonstrate the validity of the assertion by analyzing the mechanism that enforces it. Enforcement of the least-privilege principle is an important reason to use multiple domains for TCB execution.

HARDWARE REQUIREMENTS

"It [the TCB] shall make effective use of available hardware to separate those elements [of the TCB] that are protection-critical from those that are not." [SA]

"Features in hardware, such as segmentation, shall be used to support logically distinct storage objects with separate attributes (namely: readable, writeable)." [SA]

This is another appearance of the distinction between more and less critical components of the TCB; clearly, those components that make up the reference monitor are more protection-critical, but there may be other protection-critical components as well.

Using hardware mechanisms to separate TCB components is one part of implementing the least-privilege principle. Not all TCB "domains" need be established by the same hardware mechanism. For example, part of a TCB might be defined as the code that executes with hardware-defined privilege (the "kernel," usually), and other parts of the TCB might be otherwise ordinary processes that are distinguished only by software-defined privilege (the "trusted processes"). Even though the process isolation used for the latter part of the TCB is not a hardware protection mechanism as such, it still represents use of hardware features to isolate parts of the TCB.

DESCRIPTIVE TOP-LEVEL SPECIFICATION REQUIREMENTS

"The user interface to the TCB shall be completely defined and all elements of the TCB identified." [SA]

"A descriptive top-level specification (DTLS) of the TCB shall be maintained that completely and accurately describes the TCB in terms of exceptions, error messages, and effects." [DS&V]

"It [the DTLS] shall be shown to be an accurate description of the TCB interface." [DS&V]

"The descriptive top-level specification (DTLS) shall be shown to be an accurate description of the TCB interface." [DD]

"Testing shall demonstrate that the TCB is consistent with the descriptive top-level specification." [ST]

"The procedures for examining and maintaining the audit files as well as detailed audit record structure for each type of audit event shall be given." [TFM]

"During development and maintenance of the TCB, a configuration management system shall be in place that maintains control of changes to the descriptive top-level specification, ..." [CM]

The DTLS deals with the interface presented by the TCB to ordinary users, operators, and administrators. It must be a complete description of the interface (or interfaces), and must include all ways in which a user can interact with the TCB. This may actually be easier to define than the TCB boundary (see TCB Isolation, above), since it deals only with correct, expected TCB operations, rather than all potential actions. The DTLS need not be packaged as a single document, but all its components should be readily identifiable.

One of the "effects" of the TCB interface is the generation of audit messages. This is part of the DTLS, and as such must be documented and tested; in this case, however, it is acceptable to consider that portion of the TFM documentation as also a part of the DTLS (or vice-versa).

CONFIGURATION MANAGEMENT REQUIREMENTS

"During development and maintenance of the TCB, a configuration management system shall be in place that maintains control of changes to the descriptive top-level specification, other design data, implementation documentation, source code, the running version of the object code, and test fixtures and documentation." [CM]

"The configuration management system shall assure a consistent mapping among all documentation and code associated with the current version of the TCB." [CM]

"Tools shall be provided for generation of a new version of the TCB from source code." [CM]

"Also available shall be tools for comparing a newly generated version with the previous TCB version in order to ascertain that only the intended changes have been made in the code that will actually be used as the new version of the TCB." [CM]

"The procedures for secure generation of a new version of the TCB from source after modification of any modules in the TCB shall be described." [TFM]

The object of the configuration management requirements is the continual assurance of correct system design and implementation. Although these are not directly related to system architecture per se, they do have strong bearing on the maintenance of that architecture and the preservation of its "inherent" security properties.

Configuration management requires both a mechanism for maintaining the TCB and all TCB-related material (first excerpt) and a set of procedures (second excerpt) for guaranteeing proper correspondence among these materials. Tools and procedures for modification and validation of the TCB are required. It is not necessary for the TCB to be customer-modifiable, though appropriate tools must be available if it is. It is satisfactory for the "procedure" for securely generating a new TCB to be purchasing another version from the manufacturer.

COVERT CHANNEL REQUIREMENTS

- “The system developer shall conduct a thorough search for covert storage channels and make a determination (either by actual measurement or by engineering estimation) of the maximum bandwidth of each identified channel.” [CCA]
- “The TCB shall be able to audit the identified events that may be used in the exploitation of covert storage channels.” [AUDIT]
- “All auditable events that may be used in the exploitation of known covert storage channels shall be identified.” [DD]
- “The bandwidths of known covert storage channels, the use of which is not detectable by the auditing mechanism, shall be provided.” [DD]
- “This [design] documentation shall also present the results of the covert channels analysis and the tradeoffs involved in restricting the channels.” [DD]
- “It [the test documentation] shall include results of testing the effectiveness of the methods used to reduce covert channel bandwidths.” [TD]

The covert channel requirements address the identification, suppression, and documentation of all covert channels. In order to perform an analysis at all, however, it is necessary for the TCB to be sufficiently well-structured that all shared resources can be identified and considered as potential information flow paths.

FORMAL MODEL REQUIREMENTS

- “A formal model of the security policy supported by the TCB shall be maintained over the life cycle of the ADP system that is proven consistent with its axioms.” [DS&V]
- “A formal description of the security model enforced by the TCB shall be available and proven that it is sufficient to enforce the security policy.” [DD]
- “The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model.” [DD]
- “During development and maintenance of the TCB, a configuration management system shall be in place that maintains control of changes to the [DTLS], other design data, . . .” [CM]

This requires first that a formal security model (such as Bell and La Padula) be identified and accepted. The model must then be interpreted, identifying all the subjects, objects, operations, permissions, and mechanisms in the TCB, and showing how they correspond to the terms of the model. Unlike the model itself, which (being abstract) will usually remain unchanged during the life of the system, this interpretation must be maintained and updated as changes are made to the TCB interface.

TESTING REQUIREMENTS

- “Documentation shall describe how the TCB is structured to facilitate testing . . .” [DD]
- “The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation.” [ST]
- “Testing shall demonstrate that the TCB is consistent with the descriptive top-level specification.” [ST]
- “The TCB shall be found [by the NCSC evaluation team] relatively resistant to penetration.” [ST]

Testing is intended to show that the TCB functions correctly, is described completely and correctly by its DTLS, and is resistant to attack. Additionally, the TCB must be structured so that test cases can be constructed easily and to allow straightforward arguments for the completeness of the

test suite. All TCB functions described in the DTLS (both user interfaces and administrative or operator interfaces) must be tested by the test suite. It is important for the test suite to be internally consistent; that is, for similar functions to be tested in similar ways. It is also important for the test suite to be well-structured; as much as possible, its organization should follow that of the TCB itself, and test execution should be as automatic as possible. The assurance provided by a test suite depends entirely on how easily one can determine that the tests are complete and correct.

REFERENCES

- [Anderson72]
Anderson, J. P. *Computer Security Technology Planning Study*, ESD-TR-73-51, vol. I, AD-758 206, ESD/AFSC, Hanscom AFB, Bedford, Massachusetts., October 1972.
- [ATT86]
System V Interface Definition, Volumes I and II, 307-127, AT&T, Indianapolis, Indiana, 1986.
- [Bach86]
Bach, Maurice J. *The Design of the UNIX Operating System*, Prentice Hall, Englewood Cliffs, New Jersey, 1986.
- [Belady71]
Belady, L.A. and Lehman, M.M. *Programming System Dynamics or the Metadynamics of Systems in Maintenance and Growth*, IBM Report No. RC 3546, 1971.
- [Britton81]
Britton, K.H. and Parnas, D.L. *A-7E Software Module Guide*, NRL Memorandum Report 4702, Naval Research Laboratory, Washington, D.C., 1981.
- [DOD85]
Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, Department of Defense, Washington, D.C., December 1985.
- [Stevens74]
Stevens, W., Myers, G. and Constantine, L. *Structured Design*, *IBM Systems Journal*, Vol. 13, No. 2, May 1974, pp. 115-139.

THE SECURE DATA NETWORK SYSTEM: AN OVERVIEW

BY: Gary L. Tater
Edmund G. Kerut

BACKGROUND

In August 1986, the National Security Agency, the National Bureau of Standards, the Defense Communications Agency, and twelve communications and computer corporations initiated a special project called the Secure Data Network System (SDNS). This innovative research program focuses on designing the next generation of secure computer communications network and product specifications to be implemented for applications with public and private data networks. This paper will address the rationale and programmatic decisions for the SDNS project. The next four papers cover details of the actual architecture, services, protocols, and products.

INTRODUCTION

The explosive and unprecedented growth of computer-generated information in the free world, accompanied by rapid advances in telecommunications and data processing technology, has ushered in the Information Age in our society. The 1980s have seen this virtual explosion in the volume of information processed through public and government communications and computer systems. Unfortunately, this growth has not been met with a commensurate increase in the application of Information Security (INFOSEC) countermeasures to protect data communications. The Soviets and other nations, as well as terrorist and criminal elements have the capability to exploit this lack of security to the detriment of U.S. national interests. Exploitation of our communications by other countries may threaten the advantage of a U.S. industrial high technology base which has traditionally given us the competitive edge needed to succeed in international world markets. In a free society, it is impossible to control the flow of information -- even information which individually or collectively could have an adverse impact on the national interest.

There has been a long standing and effective partnership between the Government and private industry in meeting the national security needs of the United States. To implement its national security-related policies, the Government relies on industry for research and development, design, testing, manufacturing, installation, and often operation or maintenance of communications systems. Within the framework of the SDNS Program, Government and industry are joining resources and expertise to make available in the marketplace a large selection of INFOSEC products for protecting both classified and unclassified information for the DoD, civil, and private sector. The program goal is to make INFOSEC, by virtue of economics and transparency, attractive to a

large potential user base. With the proliferation of information security, it may be possible to successfully deprive our adversaries and unauthorized entities of our valuable information resources.

PROGRAM GOALS

As a result, the basic problem becomes one of finding cost-effective approaches to adding security to existing communications systems and networks. The major thrust of our SDNS strategy is to assist and encourage industry in developing a wide variety of INFOSEC products and systems to be made available in the marketplace at a cost and level of user friendliness to equally encourage widespread use of these products. To implement this general strategy, we agreed on four specific objectives. The first was that the companies involved early in the program would be creating specifications that could be used eventually by all companies developing products in NSA's Commercial COMSEC Endorsement Program (CCEP). A second decision was to make use of the International Standard Organization's Open Systems Interconnection (OSI) model and to concentrate our efforts on the emerging OSI protocols. Since one of the objectives of the OSI reference model is to permit the interconnection of heterogeneous computer communications systems, it is a natural choice for our selection to permit secure interconnection of communication systems that already have achieved communications interoperability. The third goal was to develop a complete security architecture for the link, network, transport, and application layers of the OSI reference model. The fourth goal was to demonstrate the feasibility of the technology and the cost effectiveness of the concepts.

SYSTEM CAPABILITY

SDNS describes the environment within which designers may provide users with the capability of transmitting data securely over a variety of communications networks. A user, in conjunction with a cognizant security administrator, can specify requirements from a range of security services and levels of assurance. Security policies are enforced by the system components and along with doctrine provide the level of assurance required for the specific environment.

Confidentiality of communications is assured by the use of government provided, high-grade, cryptographic algorithms for data encryption and traffic key generation. Adherence to applicable INFOSEC doctrine, criteria, guidelines, and good engineering practices in the design and implementation of the secure communication components will

assure a successful security evaluation and subsequent endorsement of the products. State-of-the-art key management techniques will minimize the burden associated with generation, distribution, accounting, and control of classified key in physical form. Key material required for initializing SDNS products will be centrally generated and distributed to users. Once initialized, secure communications components will independently establish traffic encryption keys over the network. The key management components will also support electronic rekeying of the secure communications components and the process of compromise reporting, evaluation, and recovery.

The SDNS concept requires interoperability of secure communications components supplied by multiple vendors if the same services are implemented at the same protocol layer. Protocol specifications will ensure that interoperability of supplied SDNS services exists.

Each vendor can provide security features beyond the minimum required set to be incorporated in an SDNS product. Secure communications components may exist as stand-alone interface devices or may be embedded into communications or information processing equipments or systems by vendors based on their perception of user needs. SDNS will not preclude the users selecting various hosts, communications networks, and communications services for security implementation. This permits a wide range of security products certified under the auspices of SDNS, but otherwise tightly coupled to and integrated with a particular host architecture, communications network or communications service if they conform to the OSI architecture.

Two types of SDNS equipment will be designed: the first type (Type I) will be aimed at Government classified and sensitive unclassified national security information, and the second type (Type II) will be used for unclassified applications in the Government as well as in the private sector. Users and system managers will be able to specify their communications and security needs. It will be possible for users to select security services dependent upon the application, communications services that are needed, and the degree of interoperability desired. We expect that there will be a number of SDNS products tailored to specific communications and security service requirements. Components providing a common set of services will interoperate when SDNS protocols, requirements, and specifications are satisfied.

There will also be an SDNS infrastructure that contains a documented body of knowledge that will be needed by the people who design, build, operate, and use SDNS. Doctrine, procedures, guidelines, and specifications that document security management functions and activities are included in the infrastructure.

PROGRAM IMPLEMENTATION

Since August 1986, the SDNS Program has been progressing under a three-phase approach. The first phase included the development of the overall concepts, services, architecture, and key management techniques. Because of the large number of Government and industry participants, several working groups were formed to concentrate expertise on specific problems. The Protocol Working Group focused on defining the services and the protocols. The Access Control Working Group developed a methodology for distributing access control approval at the communication end points. Using the DoD Trusted Computer System Evaluation Criteria known as the "Orange Book", the INFOSEC Working Group studied the SDNS concepts and protocols to develop the appropriate criteria for which the SDNS devices will be evaluated against. The Key Management Working Group defined the requirements for the public key based system that SDNS will use.

For this first phase, in addition to the participants from NBS and DCA, NSA awarded contracts to Analytics; AT&T; Bolt Beranek and Newman (BBN); Digital Equipment Corporation (DEC); GTE; Honeywell; Hughes; IBM; Motorola; Unisys; Wang; and Xerox. In Phase I, the first task was the development of a broad communications security architecture, i.e., the set of guidelines, constraints, and rules to implement secure communications over public and private data communications networks. A range of threats to data communications, a range of environments that the architecture will be applied, and a general communications architecture were all factors in the development of the security architecture. The OSI reference model is the communications model used to establish a framework for coordinating the development of the security architecture services and related elements, which would be applied appropriately in the circumstances for which protection is required.

After definition of the architecture and services, the Protocol Working Group emphasized the development of end-to-end encryption protocols at layers 3 and 4 as well as electronic mail services compatible with X.400 at layer 7. Because a market exists for SDNS products at layers 2, 3, and 4, we decided that a key management protocol that could service these three layers was needed. This led to the definition of a key management protocol as an applications entity existing at layer 7 in the OSI model.

As of August 1987, the first phase of the SDNS program has been completed. The architecture and protocol specifications have been drafted as have been the key management and access control planning documents. The program is now in a development, testing, and validation phase and is beginning to focus on writing the protocols and developing the INFOSEC products that will merge COMPUSEC and COMSEC technology.

Since the protocol specifications will eventually be made available to vendors developing INFOSEC products under NSA's CCEP, this phase will serve to test both the specifications themselves and the operational characteristics of the security protocols on communications networks. The original concept of combining security protocols with off-the-shelf network and transport protocols will be proven. An added benefit to be gained from this phase is the demonstration to both potential vendors and users that interoperability of multi-sourced INFOSEC products is possible.

During the communications testing and interoperability testing to be conducted next year, we expect to add substantial detail to the protocol and interoperability specifications. This is expected to make it easier for companies that follow to build interoperable hardware.

CONCLUSION

While the SDNS Program has recently completed its first phase, it is still a research project several years from providing actual hardware capable of protecting our nation's data information. It has, however, offered twelve companies an opportunity to work together, and with DCA, NBS, and NSA, to influence the next generation of INFOSEC products. With the rapid proliferation of data communications and the ease of access into networks that are growing daily, we must preserve our military, scientific, and technological edge. We must take whatever steps are necessary -- before a system becomes operational -- to provide the United States the means for ubiquitous data security that is so badly needed.

SDNS SERVICES AND ARCHITECTURE

Ruth Nelson
Electronic Defense Communications Directorate
GTE Government Systems Corporation
77 A Street
Needham, MA 02194

ABSTRACT

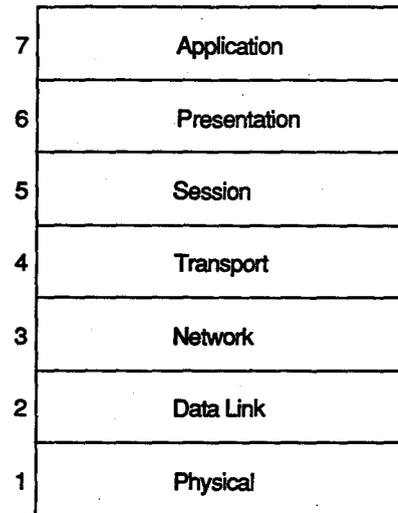
The Secure Data Network System (SDNS) includes both support for secure communications between users and a key management capability. The major elements in the system are the Key Management Center (KMC) and the SDNS terminals, which can be intelligent terminals, workstations or host computers. The SDNS architecture is consistent with the ISO OSI communications architecture and protocols as well as with the DoD protocol suite. As presently defined, it provides security services at four layers of the OSI architecture -- link layer, network layer, transport layer, and application layer (for electronic mail). The SDNS program has developed standards for network, transport and electronic mail protection; link layer standards have been deferred as not critical for interoperability across the network. In addition, protocols have been defined for key management, including communication of access control information and negotiation of communications protocol parameters.

INTRODUCTION

The Secure Data Network System (SDNS) is intended to provide secure data communications services to a variety of DoD and commercial users. These services include key management and system management capability as well as the encryption, authentication, and access control of user data. During the concept definition phase personnel from eleven contractors, NSA, NBS, and other government agencies participated in determining the security services to be offered, the system architecture, system management and access control requirements and mechanisms, and the key management and secure communications protocols. This paper will focus on the protocols and system architecture.

SDNS provides security compatible with the International Standards Organization (ISO) Reference Model for Open Systems Interconnection (OSI)¹. Figure 1 shows the OSI protocol architecture. SDNS terminals and the SDNS Key Management System (KMS) communicate using OSI protocols. Terminals may be connected to each other and to the KMS through local area networks, public or private data networks or telephone links, as shown in Figure 2. The secure communications protocols offer encryption services at application, transport, network or link layers; the key management and system management protocols utilize the OSI management approach and are application layer protocols between management application

entities. Figure 3 shows the placement of the SDNS protocols within the OSI framework. The security services at each layer are a subset of the services described in the Security Addendum to the OSI Reference Model².



The OSI Protocol Architecture

Figure 1

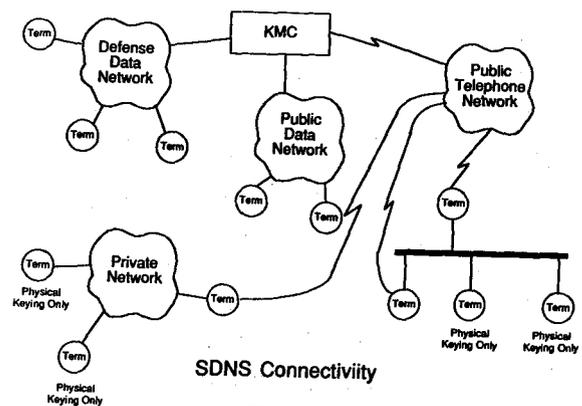
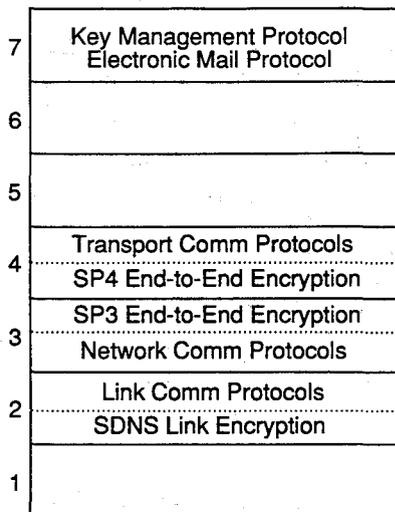


Figure 2



Placement of SDNS Protocols

Figure 3

KEY MANAGEMENT

The heart of SDSN is the Firefly keying system. Each terminal has a unique Firefly key which is bound together with a non-forgeable certificate. The certificate identifies the terminal and specifies its security-relevant characteristics. Two SDNS terminals desiring to communicate exchange certificates and keying information (the Firefly exchange) and make access control decisions based on the identifying information. The exchange generates a traffic key which is unique to the two terminals and which is new for that key exchange. If communication is permissible, the terminals then negotiate the communications parameters for use of the traffic key.

The Firefly keys and certificates are issued by the Key Management Center, which receives key orders and maintains accounting information, but is involved in neither the terminal to terminal communication nor formation of traffic keys. Initial delivery of Firefly keying material from the KMC is physical and consists of either operational key or seed key. These are similar except that seed key can only be used for connecting to the KMC for conversion to operational key. Operational keys are classified at the level of the traffic which they protect; seed keys are always unclassified. The KMC also provides an electronic rekey service for terminal operational keys.

Besides the real-time Firefly exchange between the terminals, the system also provides a mechanism whereby a terminal can provide Firefly keying information ahead of

time (for example, by posting its certificate and keying information on a bulletin board). This can be used by a second terminal to generate a traffic key for secure electronic mail.

The protocol working group for SDNS has defined a Key Management Protocol (KMP) which is used for seed key conversion, electronic rekey, real-time Firefly exchange, and update of terminal traffic keys. This is an application-layer protocol, designed to be compatible with the ISO OSI Management Services architecture. The communications are between Key Management Application Entities (KMAE) in the terminals and in the KMC. Each KMP transaction includes a Firefly exchange between the KMAEs to establish a traffic key and then a secure exchange of security-services data using that traffic key. The successful use of the traffic key validates the identities of the communicating devices and tests the key. The exchange of security services data in the real-time key-exchange transaction is used to convey additional access control-related information and to determine the parameters for use of the traffic key in encrypting user data. The same KMP exchange is used for generating and validating traffic keys for the network layer and transport layer encryption protocols currently defined for SDNS; it will also be used for link encryption and other real-time encryption protocols when they are defined. The KMP protocol does not support secure electronic mail keys; these are generated by a different, non-real-time Firefly exchange, described in the section of this paper on electronic mail.

Application layer key management protocols allow use of a common set of management protocols across all SDNS devices, independent of which user services the devices provide. They also allow the system to evolve and device manufacturers to add new user services without impacting their interoperability with the key management system.

END-TO-END ENCRYPTION

In packet networks and internets, the term end-to-end encryption has been used to refer to an encryption scheme that encrypts user data but provides unencrypted network headers so that the data can be delivered through the packet network. This type of encryption must be above the network protocol in the hierarchy, since the network headers are not encrypted. In order to provide encryption service in a uniform manner to a variety of applications, it should be as low as possible in the protocol hierarchy. This kind of argument has led to encryption protocols at the top of the network layer (at the internet layer in the DoD architecture) and at the transport layer of protocol. There is not yet agreement among network security experts as to which of these is the more appropriate choice. DoD projects have primarily focused on the internet layer; ISO and commercial efforts have been at the transport layer. The Security Addendum to the OSI Reference Model allows either choice.

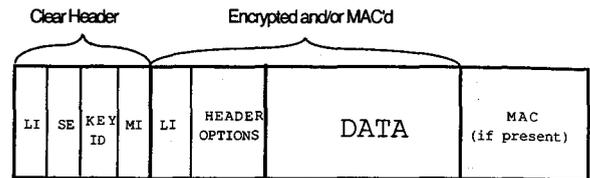
SDNS has defined network and transport encryption protocols which are consistent with each other in format and in basic services. This consistency will allow SDNS system developers to implement one or both of these protocols in an efficient manner. It will promote greater interoperability of SDNS systems and will also simplify the task of security evaluation of these systems. The SDNS transport protocol, SP4, is defined as an addendum to the ISO Transport Protocol. The SDNS network protocol, SP3, is defined as a sublayer of the network protocol which resides directly below the transport layer.

The SP3 and SP4 protocols have been developed as part of an ISO protocol suite. However, the layer interface between the ISO transport protocol TP4 and connectionless network protocol CLNP⁴ is similar to the interface between the DoD protocols TCP and IP, and the service definitions of CLNP and IP are almost identical. This leads to the possibility of SP3 and SP4 implementations which work with the DoD protocols. These implementations will be useful in securing the many existing systems now using TCP and IP.

SP3 and SP4 Services

The services provided by SP (a short term for either SP3 or SP4) are negotiated by the KMP protocol before the key is put into use. Both services and format are fixed for each traffic key, although SP3 and SP4 each support several options. The basic services provided by SP are confidentiality and connectionless integrity, as defined in the OSI Security Addendum; either or both services can be negotiated. In addition, because the SDNS key exchange provides pairwise keys, SP also provides source authentication of the protected data units. SP is an encapsulating protocol; it operates on a unit of data, and either encrypts it, provides an integrity check on it, or both. SP allows the option of including additional information with the data in an SP header, such as, for example, security labels or network service access point (NSAP) addresses. All such optional information is bound to the data and protected by the encryption and/or the integrity check. Figure 4 shows the SP header format.

SP operation is intended to be very simple, so as to facilitate any necessary certification. SP operates independently on each unit of data that is encrypted or decrypted. It does not include capabilities for sequencing, acknowledging or retransmitting data units, although the transport protocol associated with SP4 or above SP3 may have this capability. SP relies on the network protocol below it to provide communications service. If SP operates over a connection-oriented network protocol, it will provide the same quality of service as the network protocol, but will not guarantee the sequence integrity, since it does not protect network layer sequencing information.



LI - Length Indicator. Contains length of Clear Header in octets.

SE - PDU Type Code.

MI - Message Indicator

LI - Length Indicator for Encrypted Header in octets

Header Options - Each contains length and value

Options include:

- Security Label
- NSAP Address
- CLNP Header (SP3 only)
- Padding
- Final Sequence Number (SP4 only)
- Etc.

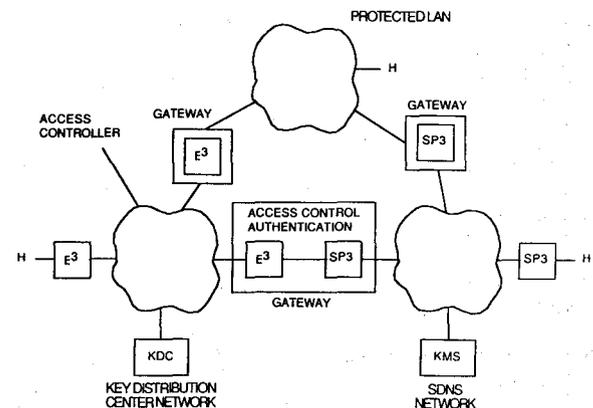
SP Header Format

Figure 4

SP3 and SP4 provide the same basic security services with the same encryption mechanisms, they operate on the same user data, and they use the same format. They can interoperate with each other if compatible options are chosen. Each protocol has been defined to include a set of compatible options, and each also includes some additional capabilities appropriate to the layer in which it operates.

SP3 Optional Services

In the OSI architecture, the transport layer is the lowest layer which is strictly end-to-end; the network layer can operate through relays (packet switches or gateways). When end-to-end encryption is done at the network layer, as in SP3, there is an option of terminating the encryption at a gateway, allowing users of a local area network to share the cost of encryption devices. Gateway encryption devices also allow interoperation through the gateway of users with different encryption systems if they use the same communications protocols (see Figure 5).

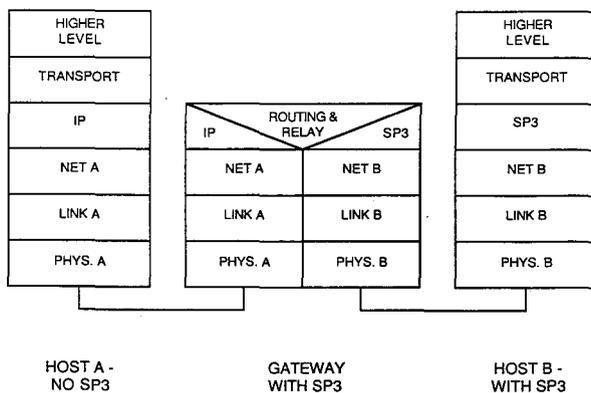


Secure Interconnection of Differing Protection Schemes

Figure 5

In order to support gateway encryption, there must be a means of routing the encrypted data through the non-secure network to the gateway which has the encryption key, as well as a means of routing unencrypted data to the correct gateway for encryption. In an internetwork with few gateways, the routing may be implicit in the host address, but in general, there can be several gateways into the same network. In addition, since the key at a gateway authenticates the gateway but not each host computer on the network behind the gateway, there must be some means other than the Firefly exchange to provide source authentication.

SP3 has the capability, if this service is negotiated by KMP, of including source and destination address information in its protected header. If the destination host is on a network behind an SP3 gateway, the unencrypted (black) network header indicates the destination address of the decryption gateway, and the destination host address in the SP3 header allows the gateway to forward the data correctly after decryption. Similarly, the protected address allows an encrypting gateway to indicate the actual source of the data in the SP3 header and its own address in the black network header. SP3 gateways can convert between SP3 with a black network protocol and a local protocol without encryption, as shown in Figure 6.



Notes: Other Network Relays May Be Present.
Net B May Actually Be An Internet.

SP3 and IP

Figure 6

If it is necessary to carry more network protocol information across a black network for use in a destination red network, SP3 offers the option of protecting an entire Connectionless Network Protocol (CLNP) header. This option simplifies the operation of the SP3 gateways, but requires SP3 protocols at both gateways and hosts to include the CLNP functionality.

SP4 Optional Services

SP4 is a part of the transport protocol, TP. Because of this, it has access to transport protocol information, such as sequence numbers and connection open requests. SP4 has defined some optional services which take advantage of its position in the transport layer. Both SP3 and SP4 allow various keying granularities, including per pair of SDNS entities and per NSAP pair. SP4 also allows a key per transport connection, which ties the protocol data units to a specific connection identifier. When used with the TP4 protocol, SP4 can provide data stream integrity. The integrity service uses the transport sequence numbers, which are protected by SP4 and an added mechanism for gracefully closing the connection.

ELECTRONIC MAIL ENCRYPTION

The SDNS protocol for secure electronic mail is an extension of CCITT recommendation X.400⁵. This standard defines the electronic mail service provided by two entities: a mail user agent (UA) and a mail transfer agent (MTA). The user agent acts on behalf of a particular user and allows him or her to generate and receive mail. The mail transfer agent is responsible for getting the mail through the network from user agent to user agent. MTAs can reside in the same computer systems as UAs or they can reside in mail relays. Their function is analogous to the post office. The SDNS protocol is at the UA sublayer and assumes that MTAs may be untrusted. Mail remains encrypted from user agent to user agent, through any relays.

The real-time Firefly exchange is not used for electronic mail. The SDNS mail protocol uses a staged Firefly exchange in which a user who wishes to receive secure mail posts his certificate and some public keying information on a bulletin board or gives it to a sender in some other way. The sender uses the posted information along with his own private information to construct a traffic key, which is unique to the sender-receiver pair.

The protocol accommodates multiple addressees by using a single message key to encrypt the message and then using a pairwise key for each addressee to encrypt both the message key and an integrity function of the message. The message header includes the sender's certificate. It also includes, for each addressee, the public keying information needed to construct his pairwise key, and the encrypted key and integrity check word. Privacy of the message is protected because only the correct recipients can construct the pairwise keys and decrypt the message key. The source of the message is authenticated by the binding between the sender's certificate and each pairwise key.

The electronic mail protocol includes an optional electronic signature. This will provide the additional service of non-repudiation, as distinct from source authentication, allowing the receiver to prove the identity of a sender to a third party.

FUTURE SERVICES

The concept definition phase of SDNS has concentrated on defining interoperability requirements for the key management services, for end-to-end encryption and for electronic mail. Once the protocols for these functions are implemented and tested, it is likely that additional functionality will be standardized and provided. Link encryption is the most often used encryption method and can be included within SDNS in a reasonably straightforward manner. Initial standardization was deferred primarily because of the multiplicity of communications protocols at the link and physical layers, but also because link encryption is better understood than either end-to-end or electronic mail encryption. Application or presentation layer encryption for real-time data transfer is another likely development. The encapsulation techniques already designed for mail, end-to-end-encryption and the KMP service exchanges can be a useful model for file encryption. SDNS already allows connection over a variety of public data networks, local area networks and the telephone network; this variety is likely to increase and perhaps to include ISDN. All of these expanded capabilities are consistent with the current SDNS architecture and the layered OSI protocol approach.

ACKNOWLEDGEMENTS

The architecture and protocols described in this paper were developed by the SDNS Protocols and Signalling Working Group during the Concept Definition Phase of the program. The members of this group represented the ten SDNS terminal contractors, GTE (the key management contractor), Analytics, the National Computer Security Center (NCSC) and various other government organizations. It was my privilege to chair this group. Some of the concepts for the security architecture, and particularly for end-to-end encryption, were developed by GTE under a NCSC research program on Internet Security Architecture and Protocols, whose participants included GTE, Unisys, NCSC and DCA.

REFERENCES

1. ISO 7498 Information Processing Systems - Open Systems Interconnection - Basic Reference Model.
2. ISO DP 7498/2 Information Processing Systems - Open Systems Interconnection - Security Architecture.
3. ISO DIS 8073 Information Processing Systems - Open Systems Interconnection - Transport Protocol Definition.
4. ISO DIS 8473 Information Processing Systems - Open Systems Interconnection - Protocol for Providing the Connectionless Mode Network Service.
5. CCITT Fascicle VIII.7 Data Communications Networks. Message Handling Systems Recommendations X.400-X.430.

SP4: A TRANSPORT ENCAPSULATION SECURITY PROTOCOL

Dennis Branstad, National Bureau of Standards
Joy Dorman, Digital Equipment Corporation
Russell Housley, Xerox Corporation
James Randall, International Business Machines Corporation

INTRODUCTION

The Secure Data Network System (SDNS) project is developing a security architecture within the Organization of International Standardization's (ISO) Open Systems Interconnection (OSI) computer network model[1]. The security architecture is designed to provide several security services to the user of an OSI network[2]. The architecture includes security protocols between peer entities of the OSI architecture. The SDNS architecture is designed to satisfy the security requirements of both classified and unclassified applications. The cryptographic algorithms used for data confidentiality, integrity and key distribution have been defined but are not discussed in this paper.

The SDNS project began during the summer of 1986, Phase I, completed in mid-1987, specified the security architecture. The SDNS architecture concentrates on the confidentiality, integrity, identification / authentication, and access control security services. Non-repudiation is of secondary interest. SDNS provides security services in four of the seven layers in the ISO model.

The application layer (layer 7) provides for application specific access to network services. SDNS examined the X.400 message handling system (electronic mail). SDNS secure electronic mail provides all four of the major security services and sender non-repudiation.

The physical layer (layer 1) provides a physical connection for the transmission of data by electrically encoding the data for a specific medium. The SDNS architecture provides for confidentiality at this layer. It is the only layer in the SDNS architecture which provides traffic flow confidentiality.

The network layer (layer 3) provides message routing and relaying between interconnected networks and end systems on the same network. The SDNS architecture provides all four of the major security services at this layer. Connectionless confidentiality and integrity are provided. Identification / authentication and access control are of the end systems. It is the only layer in the SDNS architecture which provides for encipherment at gateways to support "red" networks.

The transport layer (layer 4) provides reliable, transparent transfer of data between end systems. Again, SDNS provides all four of the major security services at this layer. This paper discusses these security services and protocol that implements them. The paper also outlines the requirements for key management.

The Security Protocol at Layer 4 (SP4) was developed by the SDNS Protocol Working Group. SP4 provides either connectionless or connection-oriented confidentiality depending on the cryptographic key granularity. Likewise, either connectionless or connection-oriented integrity may be selected. Peer entity authentication and access control are provided in conjunction with the key manager.

The following objectives were established in designing SP4:

- provide secure end-to-end reliable service independent of network technology
- provide confidentiality and integrity cryptographic protection continuously from one end system to another
- provide ease of implementation when red/black separation is required
- support both host-to-host keying and transport connection keying
- support many cryptographic algorithms
- support many different generic transport protocols
- minimize changes to existing transport services and protocols
- minimize the effort, cost and time required to achieve security certification for classified applications
- minimize the bandwidth of covert channels (i.e., information paths that would allow unprotected data to exit from an end system)
- allow implementation within end systems with varying levels of trust

In order to satisfy the selected set of objectives, an encapsulation approach was taken. Transport encapsulation security was coined to denote that whatever the transport entity produced to send to a peer transport entity was encapsulated in a security envelope. This new envelope, called a Secure Encapsulated Transport Protocol Data Unit, could then be sent through any network. A simple format was defined and the required security transformations were specified.

KEY MANAGEMENT SUPPORT

The keys provided by the key manager are used by SP4 to provide confidentiality and integrity. Access control and authentication decisions are made before the key identifier is delivered to SP4. SP4 enforces these access control decisions by checking the labels on individual protocol data units (PDU).

Key Generation

SP4 was designed to be independent of encryption algorithm and method of key distribution. Either symmetric or asymmetric algorithms can be used.

SDNS uses SP4 with a symmetric key algorithm. SP4 depends on the key manager to establish and update traffic keys. The SDNS key manager uses public key cryptography to generate these traffic keys.

Key Granularity

One of three key granularities is selected when the key is established:

- Key per end system NSAP pair. One key protects all transport connections established between a pair of transport entities in two end systems.
- Key per end system NSAP pair and security label. As above with the addition that the protection extends to a single security level or range.
- Key per transport connection. One key will be used to protect each transport connection independently from all others. Transport connections are assumed to be single-level. Transport connection keying is required for connection-oriented integrity.

A SP4 transport entity may simultaneously support any or all of these key granularities. Security options are associated with each key identifier; this technique permits traffic to be protected to varying levels.

Security Option Association

When one of the transport entity pair keying alternatives is selected, the following attributes may be associated a key identifier:

- Encryption algorithm
- Confidentiality (encrypt or not)
- Message Authentication Code (MAC) length (including none)
- Security label in each protocol data unit (or not)
- Set or range of security levels which may be transmitted under the key

If transport connection keying is selected, the following attributes may be associated with a key identifier:

- Encryption algorithm
- Confidentiality (encrypt or not)

- Message Authentication Code (MAC) length (including none)
- Security label in each protocol data unit (or not)
- Connection truncation protection (or not)

PROTOCOL AND DATA FORMAT

SP4 provides many security services. This section further defines these services and discusses how each is provided. SP4 relies on the key manager and generic transport services; the dependencies will be highlighted.

Protocol Data Unit Format

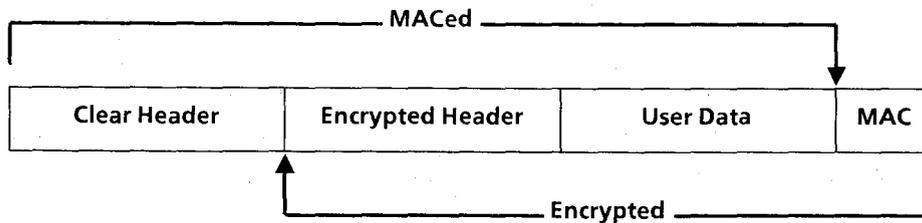
Figure 1 illustrates the format of the protocol data unit (PDU) used in SP4. The SE PDU is formed by computing the message authentication code (MAC)^[3] and then performing encryption.

Four heading fields are transmitted in the clear. The first field is the Length Indicator (LI); it simply points to the beginning of the encrypted information. Second is the type field; SP4 PDUs always have SE for their type. Next is the key Identifier (KEY-ID). The key identifier names the key; including a name permits different connections to be cryptographically separated on the network. Finally, the Initial Vector (sometimes called the MI) appears. The recipient uses the Initial Vector to initialize the decryptor; this value permits the PDUs to be decrypted even if they arrive out of order

The encrypted header also contains four fields. The Security label, Final Sequence Numbers (FSN), and Pad are optional; only those which are needed are included. The LI points to the beginning of the user data. The security label indicates the sensitivity of the data contained in the PDU. The FSN gives the final transport sequence number sent and the final transport sequence number received. The FSN is included in the closing PDUs of the transport connection. Pad is used when the encryption algorithm requires the PDU to be a specific length.

Confidentiality

Confidentiality is the protection of information from disclosure to unauthorized individuals, entities, or processes. Connectionless confidentiality is the



Clear Header =

LI	SE	Key ID	MI
----	----	--------	----

Encrypted Header =

LI	Security Label	FSN	Pad
----	----------------	-----	-----

Figure 1. SE PDU Format

protection of a individual PDUs. Connection-oriented confidentiality is the protection of all PDUs in a transport connection.

SP4 supplies connection-oriented confidentiality when transport connection keying is used. Otherwise, connectionless confidentiality is provided.

Connectionless Integrity

Data integrity is the protection of data from alteration or destruction. Connectionless integrity provides protection against the modification of a individual PDUs.

SP4 provides connectionless integrity by appending a MAC to the PDU. The MAC algorithm uses the same key as the encryptor / decryptor, so an additional KEY-ID field is not required to support the MAC. The MAC is computed on the entire PDU, including the plaintext header. The MAC is computed before encryption and checked following decryption.

Connection-oriented Integrity

Connection-oriented integrity includes protection against modification, deletion, insertion, replay (of single PDUs and entire connections) and reflection.

Protection against modification is provided as in connectionless integrity; the MAC provides this protection.

Protection against insertion is provided by the MAC and the sequence numbers of the generic transport layer. These sequence numbers are part of the encapsulated "user data".

Protection against deletion is provided by the same two facilities (MAC and transport layer sequence numbers) plus the final sequence numbers fields on the closing PDUs. The MAC and transport layer sequence numbers are sufficient to detect PDU deletions in the middle of connections. The ISO OSI Transport Protocol (TP)[4,5] is vulnerable to deletion of the end of a connection. SP4 includes the final sequence number received and sent on

the closing PDUs to detect this truncation. Truncation is not prevented; it is detected.

Protection against PDU replay is obtained if the sequence numbers do not wrap around under the connection key. SP4 must obtain a new key from the key manager should the sequence number space be exhausted.

SP4 must ensure that each transport connection is separately keyed. The key manager is responsible for performing a liveness check as part of key establishment. At connection release, SP4 must also notify the key manager to destroy the key.

Protection against reflection is provided if the KEY-ID for transmit and receipt are different. This is accomplished either by the use of different keys for the sender and the recipient or by different names for the same key.

Table 1 summarizes the division of responsibilities between generic transport, SP4 and the key manager to achieve connection-oriented integrity.

Access Control

Access control provides protection against unauthorized use of the resources accessible via OSI. Access control is provided by the key manager. In addition, SP4 provides support for access control via security label checking. SP4 discards any PDUs that arrive and decrypt but contain labels outside the range specified for use with the key identifier.

Peer Entity Authentication

Peer entity authentication is the verification that a peer entity in an association is the one claimed. This service can be provided both during the establishment of a connection and during the data transfer phase of a connection. SP4 does not provide peer entity authentication at connection establishment. This service is provided by the key manager.

Protection Against	Generic Transport	SP4	Key Manager
Modification	--	MAC	--
Deletion	Sequence numbers	MAC	--
Insertion	Sequence numbers	MAC & Final sequence numbers	--
PDU Replay	Sequence numbers	MAC & No wrap in sequence numbers	--
Connection Replay	--	--	Liveness test & Key per connection
Reflection	--	--	Different Key IDs in each direction

Table 1. Connection-oriented Integrity Division of Responsibilities

CONCLUSION

SP4 conforms to the OSI philosophy of putting desirable services in the lowest layer possible that can achieve the goals. The host-to-host nature of the transport layer, the encapsulation strategy, and the separation of the key management give SP4 security and flexibility. SP4 meets all of its design objectives.

Since the transport layer is above the network layer, SP4 passes through routers and relays untouched. This host-to-host quality, along with encryption, fulfills the following design objectives:

- provide secure end-to-end reliable service independent of network technology
- provide confidentiality and integrity cryptographic protection continuously from one end system to another

The encapsulation strategy used in SP4 permits it to use any generic transport protocol including DOD's TCP and ISO's TP. Since the encapsulation is done as the last step in the transport layer, SP4 can be implemented within the host or within the network front end processor. When SP4 is implemented in a front end processor, the security boundary becomes obvious. The encapsulation technique reduces the covert channel bandwidth by filling all of the plaintext SP4 heading fields without influence from the user. Encapsulation fulfills the following design objectives:

- provide ease of implementation when red/black separation is required
- support many different generic transport protocols
- minimize changes to existing transport services and protocols
- minimize the effort, cost and time required to achieve security certification for classified applications
- minimize the bandwidth of covert channels (i.e., information paths that would allow unprotected data to exit from an end system)
- allow implementation within end systems with varying levels of trust

Separating the key management from the SP4 protocol fulfills the remaining two objectives:

- support both host-to-host keying and transport connection keying
- support many cryptographic algorithms

REFERENCES

- [1] ISO 7498, Information Processing Systems - Open Systems Interconnection - Basic Reference Model.
- [2] ISO 7498/2, Proposed Draft Addendum to ISO 7498 on Security Architecture.
- [3] National Bureau of Standards, Data Encryption Standard, Federal Information Processing Standards Publication 46, 1977.
- [4] ISO 8072, Information Processing Systems - Open Systems Interconnection - Transport Service Definition.
- [5] ISO 8073, Information Processing Systems - Open Systems Interconnection - Transport Protocol Specification.

SDNS PRODUCTS IN THE TYPE II ENVIRONMENT

John Linn

BBN Communications Corporation
Cambridge, Massachusetts

ABSTRACT

This paper explores issues which arise in applying SDNS security technology to answer the Type II market's need to secure commercial and Government unclassified sensitive information transmitted across data networks. The Type II environment has a number of fundamental characteristics and requirements which differentiate it from the Type I (Government classified) environment. Some of these characteristics simplify issues which arise in the Type I environment, or allow enhanced functionality to be offered, but others introduce new and difficult challenges which must be addressed. This paper examines the ramifications of communications security for the Type II environment and considers the role that SDNS can play in satisfying that environment's needs.

CHARACTERIZING THE ENVIRONMENT

The potential market for Type II SDNS components can be divided into several categories: DoD unclassified users, civilian Government agencies and departments, Government contractors, and the broader private sector. In general, the non-DoD community is in a learning phase, determining needs for information security, determining communication security's role in the overall information security picture, and identifying appropriate mechanisms to answer those needs. The requirements of this emerging marketplace are still evolving.

Organizational characteristics of the Type II environment introduce new challenges for component producers. The Type II user community, particularly in the commercial sector, is not oriented to accepting security practices which interfere with operational flexibility. This places a premium on user-friendly key management. Potential customers for Type II SDNS technology use computers with a broad range of vendor-specific communications protocols. These protocols are not easily modified to satisfy a security system's needs. This suggests that security technology needs to be transparent to vendor protocol characteristics. Customers evaluate security requirements against stringent cost/benefit tradeoffs, and adoption of security mechanisms must be justified financially based on risk assessment. The use of embedded communications security (COMSEC) technology offers a significant potential to provide the cost-effective security solutions which this marketplace requires.

ISSUES AND CUSTOMER CONCERNS

Many of the basic tenets of communications security, its goals, and the context in which it operates, differ between the classified and unclassified environments. To cite a few examples:

1. Relative to the Type I environment, Type II environment definitions for clearances, sensitivity levels, sensitivity categories, and data labeling are less formalized and are

fragmented among larger numbers of administrations. Clearances assigned by one organization are not generally transferable to, or interchangeable with, those of other organizations. Similarly, sensitivity levels and category definitions are not generally interchangeable across organizational boundaries.

2. The rule-based, administratively-directed access control policy associated with the DoD clearance lattice and enforced by Orange Book A and B level hosts (mandatory access control, in TCSEC parlance) is alien to the present Type II marketplace. In particular, the commercial market's security policy needs differ significantly from classified military needs, as [Clark] has noted. No analog to the TCSEC hierarchic security levels exists across the Type II environment. In principle, the TCSEC non-hierarchic access category concept could be applied to Type II needs to segregate trade secret information, proprietary information, and the like, but even this application is complicated by the issues noted in 1. above. Therefore, it appears that most Type II access control will be identity-based, that is, making decisions as a function of the authenticated identities of would-be accessors, rather than being based on rules granting access as a function of attributes (e.g., labels) of data.
3. Data integrity is emphasized in the Type II environment, even in those contexts (such as portions of the financial community) which do not impose data confidentiality requirements. In those Type II contexts where data confidentiality is required, traffic flow confidentiality is not generally a concern.
4. In the Type II arena, administrators are not commonly concerned with the prospect that Trojan horses might leak data out of their systems into less secure environments. The absence of this concern facilitates integration of SDNS functions within host computers.

Other issues are common between the Type I and Type II environments, for example, the recognized need for authentication, data integrity, and identity-based access control services. Authentication is important not only as a prerequisite to access control, but also as a service in its own right, providing trustworthy identification data for use by host computers and users. In the financial community, in particular, there is precedent for an authentication service requirement even in the absence of a confidentiality requirement; authentication and data integrity services in this community have traditionally been provided at the application layer, independent of any confidentiality services which may be provided at lower protocol layers.

In the SDNS architecture, peer component authentication depends on attributes bound into the cryptographic keying material which is distributed to a component in order to make it

operational. The decentralized nature of authority in the Type II environment strongly suggests that the generation and dissemination of keying material be decentralized, so that users' changing attributes can be reflected in a timely fashion. It is also critical that keying material be available to Type II customers in a cost-effective manner. These issues suggest important technological and policy tradeoffs with regard to key management services for Type II customers. Acceptance of SDNS security in this marketplace is likely to depend on the successful resolution of these tradeoffs.

Two Type II environment attributes appear contradictory and this apparent contradiction deserves examination. On one hand, some Type II customers identify internal threats to their computing installations (e.g., authorized users exceeding their authorization and performing inappropriate actions within a system) as a major security concern. Despite this fact, trusted computer system technology has not received major emphasis in the Type II environment. There are several possible explanations for this apparent dichotomy. In general, the internal trust level of current commercial products is limited and has not been a selection criterion driving users to choose one product instead of another. When security features are considered, it is often on the simple basis of their presence or absence, rather than on their evaluated quality. If a facility is incorporated into a component, it is expected to operate correctly and perform its designed functions. For example, the simple presence of an access control mechanism might suffice to satisfy procurement goals, even if the mechanism's design or implementation were susceptible to malicious subversion. The TCSEC emphasizes DoD mandatory controls which lack clear applicability in the unclassified environment. This may contribute to user perceptions that internal computer system trustworthiness is an issue primarily relevant to the classified arena. Further, the TCSEC emphasizes disclosure concerns over data integrity concerns, and the latter are of primary importance to many Type II customers.

Mechanisms to protect host computers and their sensitive data from unauthorized access via network communications channels are rapidly becoming important to Type II customers, independent of the internal security level of the hosts being protected. This is especially true when easily-accessed public or shared networks are used, but the same mechanisms are often needed for private networks which carry sensitive data. SDNS COMSEC components can add security value to public or private networks, offering protection against active wiretapping (ensuring integrity of sensitive data) and passive wiretapping (ensuring confidentiality where required). Moreover, SDNS components can satisfy network-oriented access control concerns in a very strong fashion. This enforcement vehicle for a local administration's policies is particularly useful in establishing protective boundaries around hosts with limited internal COMPUSEC assurance.

Covert channel bandwidth restriction is not an important responsibility for Type II SDNS components. Attempts by authorized host users to leak information into unsecured communications facilities do not appear to be a major concern.

Provision of security services on behalf of hosts, rather than enforcement of isolation between hosts and networks, is the principal emphasis. As a specific example, it will be common for SDNS-secured Type II hosts to communicate not only with other SDNS-secured hosts, but also with unsecured hosts. This implies that Type II SDNS components must accommodate selective application of encryption, either on an address-driven basis or on request from an associated host.

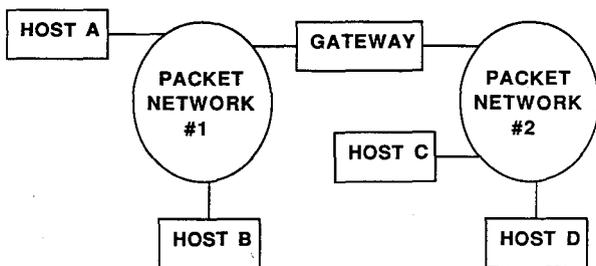
DESIGN APPROACHES

Potential customers for Type II SDNS technology use a broad range of host computers, which are supplied by a large number of vendors. In many cases these computers communicate using vendor-specific protocols at upper protocol layers, rather than ISO or DoD standards. It does not appear feasible to standardize means for integrating SDNS security measures within large numbers of upper-layer protocols specific to individual vendors. Fortunately, many of these protocols share a common denominator: they operate atop one of two standard protocols, X.25 (for long haul networks) and IEEE-802 (for local area networks). Transparent security mechanisms designed for use with these standard protocols can provide security services for a wide range of hosts, independent of the protocols employed above the layer where protection is provided. Transparency issues include performance and operational support as well as protocol compatibility. For minimal performance impact, flow control information must be reflected across SDNS components. For minimal operational impact, status information must also be reflected across SDNS components. Since covert channel bandwidth restriction is not an important concern for Type II SDNS components, it is relatively straightforward for Type II components to reflect such information and provide highly transparent service.

Security functions can be offered in the near term, using transparent security mechanisms implemented in standalone SDNS components for connection between a host computer and its network interface. An add-on SDNS end-to-end security "overlay" for a network can be offered in a non-invasive fashion, imposing minimal disruption on existing hosts' operations, as Figure 1 illustrates. Figure 1(a) shows a group of hosts, attached to a pair of packet networks which are linked by a gateway. In Figure 1(b), SDNS components are added to secure traffic among hosts A, B, and C; each of these hosts can continue unsecured communications with host D.

As the Type II marketplace's security requirements mature, it will become more likely for computer system and network component vendors to offer SDNS-based security provisions, either as optional features or standard facilities within commercially marketed systems. The use of embedded COMSEC components and modules can reduce the incremental cost of procuring security features within a computer system, assuming that cost-effective Type II CCEP components become available. The Type II environment's limited concern about Trojan horses facilitates integration of embedded COMSEC, and also facilitates the use of SDNS facilities to protect communications services at upper protocol layers.

(a) Before SDNS Security Overlay



(b) After SDNS Security Overlay

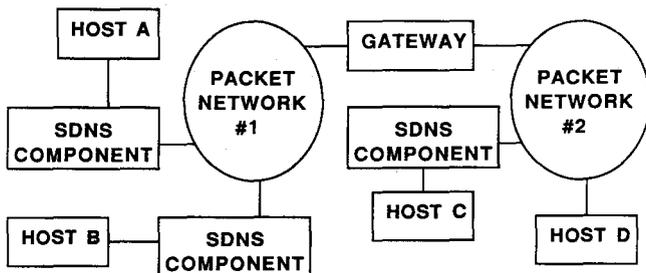


Figure 1
SDNS SECURITY OVERLAY FOR NETWORKS

Upper-layer services are difficult or impossible to protect with outboard COMSEC components interposed on interfaces between host computers and their network ports, as illustrated in Figure 2(a). Instead, it is generally necessary to integrate the COMSEC functions used to protect upper-layer traffic within a peripheral operating under host software control, as illustrated in Figure 2(b).

(a) COMSEC Interposed on Communication Interface



(b) COMSEC Integrated as Peripheral

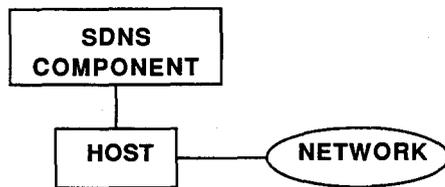


Figure 2
COMSEC INTEGRATION APPROACHES

Electronic mail is an example of an upper-layer communications service of major interest to both Type I and Type II customers. SDNS protection for electronic mail is implemented within an application layer user agent (UA) process, which corresponds to an individual human user wishing to send or receive secure mail. Electronic mail is transferred on a store-and-forward basis in which the originator's and recipient's computers need not communicate directly in real time. As a result, true end-to-end protection for this traffic cannot be achieved below the application layer. Application layer encryption implies that a large amount of control information, which is present in the headers of all seven OSI protocol layers, must be transmitted as plaintext. Type I environment concerns may dictate that measures be taken to reduce this channel's bandwidth (perhaps through use of lower layer SDNS mechanisms, in addition to application layer mechanisms). As hosts with enhanced trust levels become available, the need for such measures may diminish. In the Type II environment, however, it is reasonable to offer SDNS electronic mail security in the near term, without need for bandwidth restriction measures.

THE PATH AHEAD

SDNS technology has significant potential to provide high quality, cost-effective security for the Type II environment. To realize this potential, several important prerequisites must be satisfied, complementing the standardization activities carried out in SDNS Phase 1. Vendors and evaluators alike must focus on Type II requirements and environmental characteristics, which differ significantly from those seen in the classified arena. Inexpensive Type II CCEP modules and components, the essential building blocks for cost-effective Type II SDNS equipment, must become available. Keying material must be available to users in a cost-effective fashion. Its procedural aspects must not impose unacceptable burdens on network operations or administrative structures. If these conditions are satisfied, SDNS Type II products can offer valuable protection for unclassified sensitive and commercial data network traffic at a wide range of protocol layers. The market appears poised to grow rapidly if the right products become available. If the conditions are not satisfied, it is less likely that Type II SDNS products can be produced and marketed effectively, and hence less likely that a qualitative improvement in the security of a broad range of data network traffic will take place.

ACKNOWLEDGMENTS

I would like to thank Pat Cain, Miles Fidelman, Stephen T. Kent, and Ellen McDermott for reviewing this paper and providing helpful comments on its content. I would also like to thank Julie Moore and Pam Saia for their invaluable editorial assistance.

REFERENCE

[Clark] Clark, D. D. and Wilson, D. R., "A Comparison of Commercial and Military Security Policies," Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland, CA, April 1987.

ACCESS CONTROL WITHIN SDNS

by

Edward R. Sheehan
Analytics Incorporated
9821 Broken Land Parkway
Columbia, Maryland 21046

ABSTRACT

This paper addresses the subject of Access Control within the Secure Data Network System (SDNS). The fundamental elements of the Secure Data Network System are nonforgeable authentication information and unique pairwise key. Using these elements, the system provides five security services; confidentiality, data integrity, non-repudiation, authentication and access control. It is the prerogative of those who establish and implement a system's security policy to choose the granularity of access control they wish to enforce. The enforcement should be consistent with both the national and local policies governing a particular environment.

In this paper we discuss access control in the framework of the International Standards Organization's (ISO's) seven layer protocol model. Access control can occur only between corresponding peers at different endpoints. An access control model and its set of corresponding rules are discussed in the contexts of initial access authorization and continuous enforcement. Initial access authorization is accomplished through the process of Peer Access Authorization while continuous enforcement takes place by means of the Peer Access Enforcement process. Both of these processes are discussed in detail.

Security services may be provided in Layers 2, 3, 4 and 7 of the ISO protocol model. In separate sections we discuss the access control concerns at each of these layers. These discussions include the definition of requisite FIREFLY certificate and Protocol Data Unit information for access control.

1. INTRODUCTION: SDNS AUTHENTICATION AND ACCESS CONTROL

1.1 OVERVIEW

This paper, along with others published here, is based upon the developmental work accomplished within the Secure Data Network System (SDNS) program under the auspices of the C65 Special Projects Office. This paper addresses access control within SDNS and, as such, represents a consensus arrived at within the Access Control Working Group (ACWG). Other SDNS working groups, such as Key Management, Protocol and Systems Management address other major components of SDNS. These all have had a direct influence on the design features of the preliminary SDNS access control concepts that are presented here.

A variety of security services will be available through SDNS components, which will allow end-users to specify the "amount" of protection required for their sensitive classified or unclassified data. The SDNS will include mechanisms to support security policies which require a high level of assurance for mandatory and discretionary access control. Authentication and access control decisions will differ from domain to domain, based upon the security policy for a particular domain. SDNS allows for and supports this requirement. The system will be as transparent as possible, and have minimum impact on the reliability of the network.

Access Control decisions in SDNS are made by the end-users attempting to communicate. These decisions are made in the context of each end-user's own security policy. The FIREFLY mechanism is an intrinsic part of key management and distribution in SDNS and is the means by which an end-user receives information to make access control decisions. FIREFLY provides the fundamental elements of non-forgeable authentication information and pairwise unique key generation for SDNS. The SDNS Key Management System will accommodate provision for a combination of centralized and decentralized functions to provide the best security with minimum impact on user flexibility. Once initialized, the secure components autonomously can establish traffic encryption keys without further intervention. FIREFLY certificates are mutually exchanged by peer entities. Access control is enforced using the information contained in the FIREFLY certificate.

The ACWG has developed an Access Control model to provide a framework for the coordination of a standard set of authentication data and access control checks which will allow for the inter-communication between different SDNS users/systems when their national and local policies allow it. The model, described in subsequent sections of this paper, includes a mechanism for continuous access control enforcement and a four tiered mechanism for determining initial access authorization.

2. THE MODEL, PAA/PAE AND HOW IT ALL WORKS

2.1 INTRODUCTION

In this section we describe a four tiered model developed by the SDNS Access Control Working Group and the processes of Peer Access Approval (PAA) and Peer Access Enforcement (PAE). This descriptive model and its set of supporting PAA/PAE rules provide a decision process for determining access to information. The FIREFLY certificate defines inputs to the decision process. Other inputs to the decision process (e.g., tables specifying identity lists to support IBAC) are not contained in the FIREFLY certificate. The construction of the model in no way implies that all instantiations of SDNS should support it in its entirety.

The SDNS provides two processes for determining first time or continued access control. Figure 1 is a top level diagram of these two processes. The first provides access control during the opening of a cryptographic association between two peers, called Peer Access Approval (PAA). The second procedure provides for the enforcement of the cryptographic association while it is active. This process is called Peer Access Enforcement (PAE). Results obtained from the PAA process are passed to the PAE process through the Enforcement Vector (EV). The PAA and PAE processes are performed at each end of the association with each peer enforcing their respective local security policies.

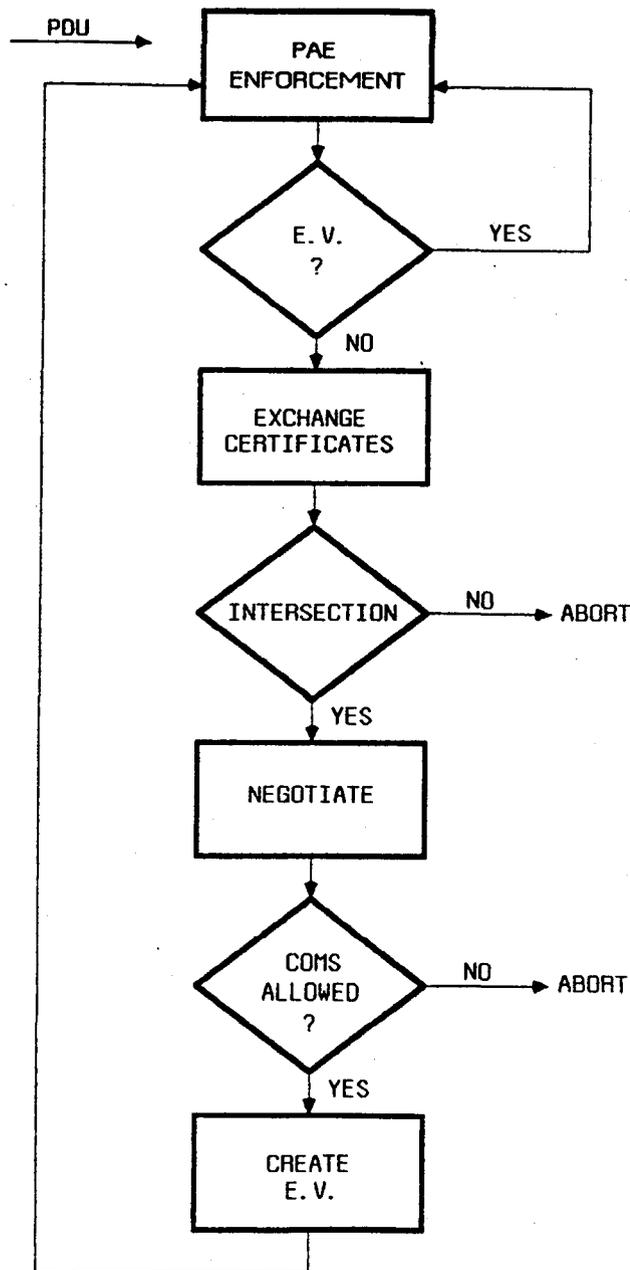


Figure 1. GENERAL MODEL

2.2 DESCRIPTION OF PAE, PAA, AND THE ACCESS CONTROL MODEL

2.2.1 Peer Access Enforcement (PAE). The Peer Access Enforcement (PAE) process is the mechanism which enforces the access control decision. The enforcement mechanism comes into play when data is sent between peer entities. The PAE is by necessity a high integrity mechanism, and is always involved in any secure exchange by virtue of acting as a permission monitor. If the PDU is an initiator for the PAA process (and the creator of an EV) then the monitor makes some preliminary checks (see Section 2.2.3) prior to starting PAA. If an association already exists then the monitor determines whether or not the labeled PDU falls within the set of permissions represented by the EV (generated by the PAA process). The PDU is either permitted to pass or not. If an association does not exist, the monitor will immediately drop into the PAA process. The PAE is not a negotiation mechanism but does perform two basic management roles; association and traffic encryption key (TEK) management.

Once an association is opened and bound by the set of permissions represented by the EV, the PAE monitors the exchanges of data to validate the access control decision parameters with each PDU. The PAE process will maintain control over the TEK for the full extent of its use, and will ensure destruction of the TEK upon the end of the cryptoperiod. In the event of a recovery procedure, the PAE process will control the TEK reuse after repeating the PAA processes for the association.

2.2.2 Peer Access Approval (PAA). Peer Access Approval is the process by which a sender/recipient uses a particular implementation of the Access Control model and its rules to determine if an association should be established.

Figure 2 is a top level diagram of the PAA process. The PAA is divided into four basic tier processes or decision making steps: global/partition, national RBAC (Rule-Based Access Control), local RBAC, and local IBAC (Identity-Based Access Control or "need to know" access control). The PAA tier processes are independent of each other and decisions to allow or disallow the association are determined and summed (logical AND) for the final PAA decision.

The PAA tier processes evaluate the elements of both peers' identity and access attributes (as presented in the FIREFLY certificate) against the local authority's security policy requirements. The PAA process also permits a security option negotiation prior to establishing an EV for an association. This negotiation could be between peer entities or through a third party. Figure 3 introduces the PAA process relative to the FIREFLY exchange and generation of the EV.

There is no predetermined order for evaluating the lower three tiers of the process. In fact, each is independent and can occur first or be omitted if security policy dictates. Tier one information is required for interpretation of some tier two and three data. The format of the data, for each tier, contained in the FIREFLY certificate will be specified. How the data is used or interpreted is left to the local authorities and their specified security policy.

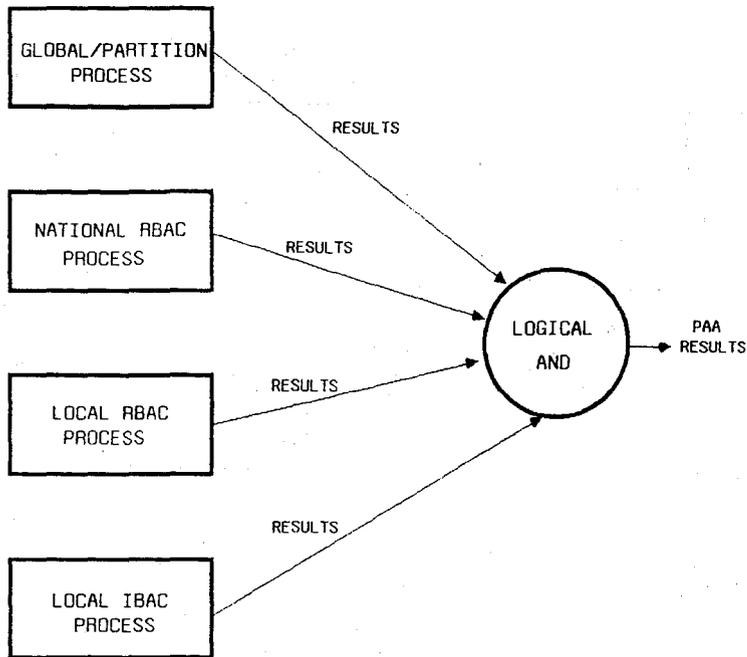


Figure 2. PAA GENERAL MODEL

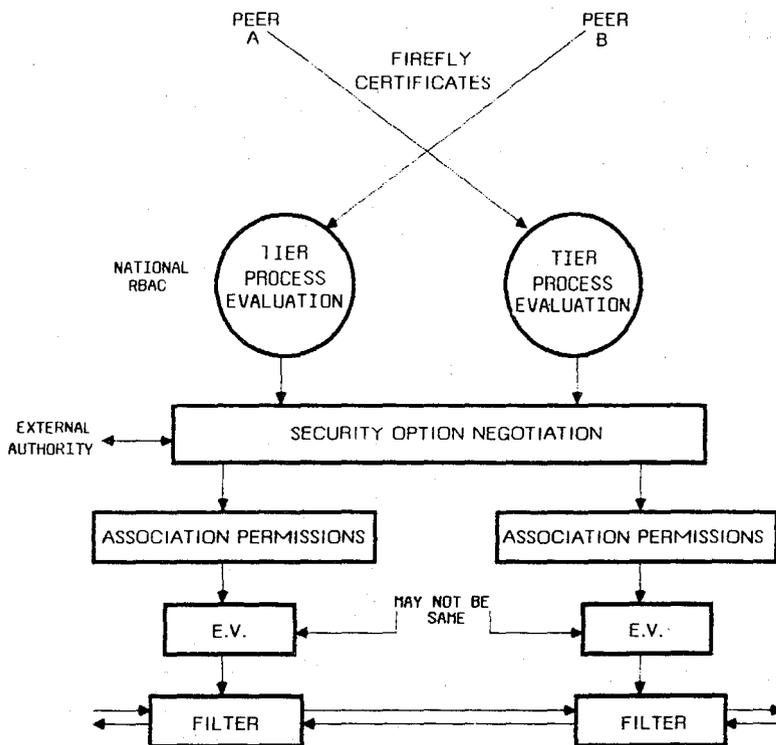


Figure 3. PAA EVALUATION

The PAA process commences with the FIREFLY exchange and ends with an access control decision to either allow the association (with the appropriate EV) or disallow the association. The resultant EV on each end is used to filter data (PAE) on a per Protocol Data Unit (PDU) basis. The results at either end could be different based upon the security policy of the local end. A simple example is one where one end employs L-RBAC while the other does not. In this case the resulting EVs may be dissimilar.

2.2.3 Access Control Model and Rules. Before the rules of the model are invoked (PAA) preliminary originator checks must be made in the context of PAE. For E-Mail first there is a check to see if the intended recipient has posted public information. In this case the PAA process (application of the rules) begins once the sender has retrieved the recipient's public data. In other cases a connection is established with the intended recipient in order to transfer FIREFLY certificates and start the PAA process. The preliminary originator checks, in non-E-Mail cases, begin when a PDU is recognized by the SDNS component. As an example, a check is made to see if an EV exists between this host pair (select key against destination address plus PAE rules). Other checks include some universal, Type I or Type II and Compromise Key List (CKL) inspection.

If it has been determined that a new key is needed for an allowable host pair the PAA process must then be initiated. Again for E-Mail the PAA process scenario is somewhat different. The sender will perform the PAA process first and the recipient will not be involved until he/she receives their mail.

The model is valid for ISO Layer 2, Layer 3/4 boundary, and Layer 7 (E-Mail), in addition to being appropriate in both the Type I and Type II worlds. An important point regarding the model is that it is not intended to dictate order. There are disjoint administrations which control the access information for their own domain. When entities wish to talk across these administrations they need to do so in accordance with some defined IBAC procedure.

2.3 INDIVIDUAL TIER DESCRIPTIONS

There are four tiers to the Access Control Model. They are briefly described in the sections which follow.

2.3.1 Global/Partition. At tier one, SDNS access control defines a mechanism which divides the population into disjoint partitions. Any SDNS device with an "active" partition cannot talk to any SDNS device with a "different" active partition number. A person/host can be a member of only one global partition per each individual FIREFLY certificate.

Once the association has been determined to have the same universal, a check at this tier will need to be made against the intersection of active partition "numbers". The results are recorded in the EV along with an approval for the association from this tier. If the partition numbers do not have a match then the process is aborted.

2.3.2 National Rule Based Access Control (N-RBAC). SDNS tier two access control is dependent upon the Global/Partition specified at tier one and is uniquely defined for that given partition. Mechanisms for separating information at this tier may or may not be hierarchical. An example of a hierarchical structure at this tier is the DoD mandatory security policy of Top Secret, Secret, Confidential, etc. Along with each of the partitions is a supporting national RBAC interconnect rule structure. (For example, permission to communicate may depend merely on the existence of a non-null intersection between the two peers.) Within a partition is a single interpretation of the national policy. Application of tier two for the Type II world is left open for further study.

The national RBAC evaluation for the hierarchical levels and/or the non-hierarchical categories results in the recording of the intersection in the EV for use in the PAE process. If a peer chooses not to enforce N-RBAC then a "don't care" is indicated by nulling out all N-RBAC fields in that peer entity's FIREFLY certificate. If a peer chooses to enforce N-RBAC and the intersection is null then communication is prohibited.

2.3.3 Local Rule Based Access Control (L-RBAC). SDNS tier three Access Control supports a mechanism to allow communities within a partition to enforce local rule-based security policies, in addition to a National RBAC. Local authorities establish the policy for the use and interpretation of local RBAC.

Compartments partition those accessing data at the local-RBAC level. These compartments may be combined with the nationally defined hierarchical security levels to determine access. As in all cases where a particular tier of the model is implemented, there must be a non-null intersection of local-RBAC compartments and appropriate security levels. If the intersection is null then the association is disallowed.

Should an intersection between peer entities exist then the information is placed in the EV for enforcement during PAE. The per-PDU information (security label) is then compared with the information in the EV.

2.3.3.1 Labeling. Since compartments (L-RBAC) as well as categories (N-RBAC) could apply across all hierarchical levels a potential ambiguity exists representing this information directly in the FIREFLY certificate. If more than one hierarchical bit is turned on it is not known whether all or some of the compartments pertain to each level. For example, Secret AB with just Top Secret is a possibility but, what could have been meant is Secret A and Top Secret B. We are currently working on understanding and documenting the labeling system in use today. Simultaneously we are trying to devise a way of representing this information in the certificate and EV without any ambiguity.

2.3.4 Local Identity Based Access Control. Tier four of SDNS Access Control allows a local authority to specify identity based controls. The FIREFLY certificate is the only source of information for SDNS IBAC decisions. Some communities may want more IBAC information than is contained in the certificate. If this is the case, there are at least two additional methods of data exchange. The first approach requires

going to a third party in realtime by one or both of the parties in the association. An example of this is using a database external to the SDNS component to determine whether the SDNS association should be allowed. The second approach is to negotiate the IBAC between the parties making the association. This approach recognizes that it may be necessary to base access control decisions on information not available within the FIREFLY certificate and, as such, may not have the same level of integrity.

3. LAYER 3/4

3.1 INTRODUCTION

At the network layer, hosts (which may, as a special case, be gateways to other networks rather than endpoints for traffic) are the peers between which SDNS access control services are applied; in other words, the subjects and objects distinguished with regard to differing access rights are hosts, not processes or individual users within hosts. The information contained in the version of FIREFLY ID defined to identify a host is appropriate as an input to this granularity of access control decision, along with layer 3 per-PDU message header information and control data structures (access control tables). Subsequent subsections will define the fields within a host FIREFLY ID, and specify relevant Layer 3 per-PDU information. Once these prerequisites are defined, the final subsections will discuss how SDNS components implement administration-imposed rule-based and identity-based access control as functions of these inputs.

3.2 ACCESS CONTROL GRANULARITY FOR LAYER 3/4

3.2.1 Secure Protocol 4 (SP4). SP4 provides communication between Transport Service Access Points (TSAPs); however, the protocol group has stated that they believe that the access control decisions would be the same in Layer 4 as in Layer 3-- they would just be performed on different objects.

SP4 is a proposed addendum to the connection and connectionless ISO Transport Protocols, DIS8073 and DIS8602. As an addendum, SP4 is not an integral part of ISO TP, but is an optional extension that provides security services. The SP4 addendum proposes a new TPDU, called Security Encapsulation or SE-TPDU, that encapsulates all other TPDU's subsequent to protocol processing functions. The SE-TPDU conforms to the structure of TPDU's specified in ISO TP DIS8073. The heading of the SE-TPDU includes a variable length label field that can be used for Access Control decisions.

3.2.2 Secure Protocol 3 (SP3). The SP3 proposal provides authentication of NSAPs instead of TSAPs. SP3 supports both RBAC and IBAC decisions. SP3 is not expected to have an impact on the surrounding protocols, so limitations of the labeling by the other protocols is not a factor. SP3 provides a security label that is independent of the underlying network protocol. The TPDU security label may be mapped into the SP3 security label if access is authorized.

3.3 ID FIELD: DEFINITION AND USAGE

The FIREFLY ID MUST provide the following information:

- o The security levels and fields to support the model in Section 2 of this document.
- o Environmental and certification information for RBAC (if enforced).
- o A unique identity for authentication.

3.4 PER-PDU INFORMATION: DEFINITION AND USAGE

Each SP3 and SP4 PDU includes a security label field and information that identifies the cryptographic association. An integrity mechanism protects all the Access Control information. The label is used to enforce a check of the security level of the data against the security range of the cryptographic association.

3.5 RULE BASED ACCESS CONTROL

When National RBAC is enforced, the intersection of the allowable security levels for non-hierarchical compartments for the connection must not be null. For L-RBAC there must be a release category in common between the two peer entities (if release categories are used).

The following rules must be followed to allow connection of hosts with different levels of certification and different classes of users and to prevent the cascading problem.

- o Hosts can be interconnected as long as the environment of the host with the highest level of trust is maintained. (It is a superset of the other host's level of trust.)
- o If the two hosts are not accredited at the same level, the higher level host must treat all data transmitted as the highest level of data that the lower level host can contain, or associate a level of trust with each packet of data. Note that this requires a trusted guard (probably human) to verify and perform the write-down back to the correct level.

3.6 IDENTITY BASED ACCESS CONTROL (IBAC)

Access Control consists of two distinct steps: 1) authentication, and 2) authorization. The FIREFLY ID and the PAA protocol provide authentication. The PAE ensures that the authentication is maintained for the entire association. The Source Address/Destination Address in the Protocol Data Unit (PDU) and the cryptographic separation provided by the algorithms are sufficient to maintain authentication at this level. Authorization can be done in many different ways in SDNS. Once the identity is provided by the PAA exchange, this identity can be used to grant rights or privileges to the host. All rights granted at this stage must be subject to the constraints of the RBAC or National Policies as mentioned.

4. LAYER 7 ELECTRONIC MAIL

4.1 INTRODUCTION

As a result of the overall scope of SDNS Phase I, the Access Control issues and mechanisms discussed in this section relate to electronic mail and are not necessarily applicable to protection of other application layer services (e.g., FTAM). Different application layer services may require different Access Control services and mechanisms. In particular, the store and forward delivery characteristic of electronic mail introduces a number of special issues which do not apply to an environment in which peer entities communicate directly in real-time. On the other hand, certain characteristics and issues discussed here are likely to be relevant to other application layer SDNS services addressed in the future: the placement of application layer peers at the top of the layered protocol hierarchy

virtually dictates that application layer SDNS functions will be integrated within a host computer or within a peripheral associated with a host computer, not in a device interposed on the computer's network interface connection. In order to support application layer SDNS functions, users must rely on processing performed within hosts.

The access control services discussed here are relevant only to the protection of electronic mail traffic between user entities. They are not applicable to the protection of control traffic passed between the internal application layer peer entities which exchange control traffic among SDNS components, or to the control traffic passed between SDNS component internal application layer entities and the Key Management System (KMS). Moreover, they are not applicable to control of access from an originator SDNS component to an intermediate black network component such as a mail or directory server. Since such controls would not involve peer SDNS components at both ends of a path, they are outside the SDNS purview.

4.2 ACCESS CONTROL AND ELECTRONIC MAIL

At the application layer, the peers between which SDNS access control services are applied are User Agent (UA) processes, corresponding to individual users, within end system hosts. SDNS functions will be integrated within the UAs, which are instantiated to support identified individual users. It is reasonable and consistent, therefore, to provide security services at per-user granularity. The information contained in the version of FIREFLY ID defined to identify a user/entity within a host is appropriate as an input to this granularity of access control decision, along with Layer 7 per-PDU message header information and control data structures (access control tables, possibly supported by servers). A subsequent subsection will specify relevant Layer 7 per-PDU information, and will discuss how control data structures are used. In composition, these mechanisms allow SDNS components to implement administration-imposed rule-based access control (RBAC) and identity-based access control (IBAC).

4.2.1 Types of Access Control Policies. Identity-based, administration-imposed E-Mail access controls incorporated in application layers SDNS modules can constrain mail dataflows in accordance with locally-defined policies (e.g., "who is user A allowed to send mail to?") Rule-based policies can also be appropriate at the E-Mail application layer. In a multi-level host with appropriate inter-user segregation mechanisms, the differing access rights of users with different clearances (as defined by the users' FIREFLY IDs) can be distinguished by application layer SDNS modules. A particular SDNS instantiation may perform neither, either, or both types of controls; where identity-based and rule-based controls are active, both sets of checks must succeed before access is granted.

4.2.2 Impact of Store-and-Forward Delivery. The store-and-forward delivery mode characteristic of E-Mail is incompatible with access control mechanisms which depend on a second exchange after the initial peer-peer exchange of FIREFLY quantities. (When network or transport layer SDNS services are employed in addition to application layer E-Mail services, post-FIREFLY exchanges may occur between the lower layer peers protecting segments of the store-and-forward path supporting E-Mail transfer, but this is independent of the application layer mechanisms discussed in this

section.) Originator and recipient UAs do not, in general, communicate in realtime; as a result, the information contained in a recipient's certificate (as posted on a server or bulletin board system) must be sufficient to allow an originator to perform any desired access control checks.

4.3 ID FIELD: DEFINITION AND USAGE

It is appropriate to consider the means by which users are identified within X.400 mail, as identifying fields in X.400 message headings must be associated with FIREFLY ID fields. X.420 notes that E-Mail users may be identified in two ways: with an Originator/Recipient (O/R) name (a construct formally defined in X.411) or with a free-form name; for universal applicability in SDNS, use of the O/R name is assumed. Three variant forms of O/R names are defined, but the latter two are intended for special purposes (support for X.121 addressing or Telex interoperability), and the first variant is clearly the appropriate choice for consideration by SDNS.

For SDNS purposes, it is proposed that a user's ID as represented in a certificate contain the following O/R name components: Country Name, Administration Domain Name, Organization Name, Organizational Unit Names (this component may be null if Organizational Unit Names are not used within the particular Organization), and Personal Name. Note that this identifies a user in terms of organizational affiliation, not mailbox address, and hence does not preclude user mobility.

4.4 PER-PDU INFORMATION: DEFINITION AND USAGE

The value of the CCITT-specified Sensitivity Indication heading field is restricted to one of three possibilities (Personal, Private, and Company Confidential). This set of options does not correspond appropriately to the hierarchic levels enforced by SDNS RBAC in a Type I environment. Therefore the SDNS proposed changes to X.420 include an additional security label field.

4.5 RULE BASED ACCESS CONTROL

4.5.1 Inter-User RBAC Issues. On initial consideration, RBAC enforcement for E-Mail appears simpler than RBAC enforcement for host-level peers at the network or transport layers, since no interconnect rules are needed to constrain communications between pairs of human users. Each user is "trusted" to process and correctly segregate information at any level up to and including his/her highest clearance. The absence of interconnect rules between human users does not imply that no RBAC mechanisms are appropriate. While it is legitimate for a TS-cleared user to send a message to a Secret-cleared user, such a message should not contain any information designated with sensitivity higher than Secret. A sending SDNS UA can collect clearance information from certificates of a message's designated recipients, can compute the intersection with the sender's privileges, and can display that information to a sending user. (Note, however, if implemented this way, this function requires that recipient's certificates be cached or collected in realtime.)

Further, the UA can verify the relation between the level provided in the certificate and the level at which the UA process is executing. For example, a single-level UA running at the TS level should not transmit mail to a recipient whose certificate indicates Secret or lower clearance, although a multi-level UA spanning the TS and Secret levels could transmit Secret mail to such

a recipient. If SDNS uses X.420 message body part definitions incorporating security labeling provisions, SDNS E-Mail components can check this intra-PDU information against the clearances of intended recipient as indicated in their certificates.

4.6 IDENTITY BASED ACCESS CONTROL

E-Mail IBAC determinations will be based on a process involving X.400 O/R names and certificate ID fields identifying users. Orange Book requirements for individual accountability (imposed at C2 levels and above) underscore the need for user identification at the granularity of named users or groups of named users.

User identification quantities can be checked against data structures constraining the set of users to whom a given user is authorized to send secure mail. It would be convenient for IBAC checking purposes if the O/R name format is used for user identification within certificates as well; if the formats are different, an endpoint UA must be able to translate between O/R name format and the form used in certificates. An originator must be able to select an appropriate recipient certificate based on the recipient O/R name in an outbound message, and a recipient must be able to select an appropriate originator certificate based on the originator O/R name in an inbound message.

As O/R names are hierarchically qualified, it seems useful to provide analogous hierarchic qualification for IBAC features. For example, it could be appropriate to grant a particular user the right to send mail to anyone in organization A (without needing to exhaustively enumerate all members of organization A), as well as to user C (and user C alone) within organization B.

5. LAYER 2

5.1 INTRODUCTION

The SDNS access control provides for automatic PAA which authorizes the association and therefore communications between peers to exist. The SDNS also provides for PAE while the connection exists. SDNS security services at OSI Layer 2 can employ these procedures to allow unattended information processing equipment to establish a communications channel and communicate according to locally accredited security policies.

The data link layer uses the raw transmission facilities provided by the physical layer and transforms it into a communications circuit that appears to be error free to the layers above. Layer 2 functions include error detection, error correction, retransmission and flow control. The data link layer may offer several different classes or qualities of service to the network layer depending on different performance and cost parameters.

5.2 ACCESS CONTROL SERVICES AT LAYER 2

The SDNS provides a wide variety of security services, at Layer 2 the SDNS can provide access control, authentication and confidentiality to the entities represented by the PLSDU. Therefore, a Layer 2 SDNS security component can deliver security services to data communications equipment at a level of granularity associated to the PLSDU. The data link layer is highly dependent on the media or physical layer, requiring differing PLSDU coding techniques and frame definitions in order to provide the Layer 2 services mentioned above. Therefore, the confidentiality services and implementations

provided by the SDNS security component must vary with the differing medias being serviced. In addition, the SDNS defines the protocols necessary to support the higher layers in order to perform the PAA process; however, the Layer 2 protocols must be media dependent and in many cases remain outside the SDNS standard.

5.2.1 Layer 2 PAA. The Peer Access Approval process binds the peers at each end of the data link with Enforcement Vectors (EV) which contain the allowable range of security attributes and identities with which each peer is allowed to communicate. The PAA also binds the TEK and the maximum allowable association lifetime into the EV. The PAA process is similar to PAA process at higher OSI layers with differences associated predominantly with what the peer entities represent. The Layer 2 PAA uses the following tier processes:

- o Global/Partition
- o National RBAC
- o Local RBAC
- o Local IBAC

As with access control security policies at other layers, the local administrations use the SDNS security mechanisms to define the policy for secure information transfer at Layer 2 as well.

The PAA process results in an EV that contains the following SDNS security attributes and identities for Layer 2.

- o Global/Partition identifier
- o Local Authority Name
- o Compartment or Levels
- o IBAC (Address or Name)

5.2.2 Layer 2 PAE. The Peer Access Enforcement is limited to the level of granularity determined by the PLSDU and will not monitor security labels at this layer. However, the PAE process must control the following functions within the security component:

- o Valid EV
- o TEK Selection
- o Cryptographic Resync
- o Cryptographic Integrity
- o TEK Timeout
- o EV Destruction

The security component for a Layer 2 device normally allows a single data link to exist and therefore a single association to exist for the PAE process to maintain. However, this restriction is unnecessary and a single SDNS security component could support link layer multiplexing.

AN OVERVIEW OF THE CANEWARE PROGRAM

Herbert L. Rogers
National Security Agency - C6
Ft. Meade, MD. 20755

1.0 INTRODUCTION -

The National Security Agency is currently involved in several programs to hasten the day when high-quality communications security can be effectively and efficiently provided for computer networking applications. A very promising development in this arena is a program called CANEWARE. It is the purpose of this paper to present an overview of CANEWARE functionality. The paper looks at CANEWARE from the point-of-view of its system capabilities; it does not probe the internals of its hardware and software. The CANEWARE program is being performed on contract with Motorola Inc., Government Electronics Group. Operational equipment will be available in 1990.

2.0 WHAT IS CANEWARE -

CANEWARE is a development program to provide high performance security services for host computers on long haul packet switched networks. CANEWARE will facilitate military grade information security (INFOSEC) by performing host-to-host encryption of packets and by enforcing both mandatory and discretionary access control policies. The principal equipment elements are the CANEWARE Front End (CFE) and the CANEWARE Control Processor (CCP). The CFE protects host data on the network by encryption, ensures that hosts send and receive data only at authorized security levels, and enforces "need-to-know" security policies on all data exchanges between hosts. The CANEWARE Control Processor maintains the "need-to-know" data base and distributes those permissions to the CFE's, performs network security audit functions, and provides centralized administrative control and monitoring. CANEWARE is targeting compatibility with the family of systems/equipment that implement the standards of the Secure Data Network System (SDNS). SDNS is a separate NSA program to specify an Open System Interface (OSI) standard architecture for a wide variety of data networks including Packet Switched Networks (PSN's) and Local Area Networks (LAN's). In an SDNS related program, NSA is developing a Key Management Center (KMC) that will generate and distribute FIREFLY-based key material. CANEWARE will utilize this facility as its source of key material and authenticated privileges.

3.0 THE APPLICATION -

CANEWARE's primary application is X.25 PSN's. The DDN Standard Service X.25 is being implemented. A modular software and hardware design allows modification to accommodate other protocols. The full range of security services can be provided for a single PSN or across a concatenation of PSN's (a catanet). DOD's Internet Protocol (IP) is fully supported. The serviced networks/catanets may be either Red or

Black. For example, the "host" may be a gateway to a Red LAN. CANEWARE will support network applications with up to 16 Domains per system; where a Domain is a community of users that is served by a single CCP (or a redundant pair). Each CFE can retain a data base on up to 1000 crypto connections (i.e keys, address information, security levels, etc.).

4.0 FUNCTIONALITY -

The principal security services that CANEWARE provides are:

- * Encryption/decryption of all user data
- * Mandatory/Discretionary Access Control
- * Authentication of all hosts
- * FIREFLY key management
- * Multi-level security

These, and other functions, are summarized in the following paragraphs:

Host-To-Host Encryption-CFE's provide host-to-host encryption of data between authorized host pairs. Traffic encryption keys are generated via a FIREFLY exchange and shared only by a pair of communicating hosts. Encryption is by an approved algorithm implemented in compliance with the requirements of NSA's Functional Security Requirements Specification. Tempest, Security Fault Analysis (SFA), and Anti-Tamper specifications are satisfied. CANEWARE is being designed to be COMPUSEC certified at the B2 level of the "Orange Book". This requires a Formal Security Policy Model and the development of a Trusted Computing Base (TCB).

Key Management - An outstanding feature of the CANEWARE approach is that it is offering the first implementation of FIREFLY key management techniques for data networks. This is the approach being promoted by the SDNS initiatives. FIREFLY evolved from public key technology and is used to establish pair-wise traffic encryption keys for the subsequent encryption of data. FIREFLY key material will be obtained from an external Key Management Center (KMC), which NSA will establish to provide keying material for future secure data networks. The FIREFLY material obtained from the KMC will contain the identification of the CFE and that of its attached host. It will also include a listing of security levels, compartments, and other privileges authorized for the host. The CFE-to-CFE FIREFLY exchange will provide a mutually unforgeable identification and an enumeration of security clearances between communicating CFE's. FIREFLY keying material will be ordered through controlled

official channels and will be delivered to a CFE either electronically, over the network, or physically in a Data Key device. This CFE-to-CFE FIREFLY exchange accomplishes decentralized traffic key generation and access authorization; allowing continued secure operation of the network during the contingency mode when a central administrative/control node is out-of-service or is unreachable. Each individual CFE in the system will "keep book" on up to 1000 permitted crypto-connections so that it need not go to the CCP for permission, or it need not execute a CFE-CFE FIREFLY exchange, for connections previously authorized during the same crypto period.

Access Control- CANEWARE's Trusted Computing Base rigorously enforces Mandatory Access Control (MAC) to ensure that data passed from host-to-host will be within the security range, and compliant to the compartmentalization permitted, for that particular host-pair communication. MAC credentials will be included in each CFE's FIREFLY vector set and will be reciprocally transferred during the CFE-to-CFE FIREFLY exchange. MAC will support 8 security levels and over 100 compartments. The MAC's will be enforced by the CFE's, be operative in both the normal and contingency modes, and can not be overridden. Discretionary Access Control (DAC) privileges will be managed and distributed by the CCP. DAC may further constrain MAC but may not upgrade the mandatory limits. DAC will specify which CFE pairs can intercommunicate and which pairs cannot. Again enforcement is a CFE responsibility. Each CFE will maintain an include/exclude list of possible connections. The CCP will update the list when appropriate. Only when an addressee is not on either list will a CFE need to request permission to open a connection. In a contingency situation (when the CCP or its alternate is unreachable) the CFE will comply with the resident include/exclude list; new connections (those with no list entry) will be permitted but tagged for later reporting to the CCP which might explicitly revoke intercommunication (contingency behavior is also a MAC parameter). CANEWARE will authenticate the CFE-to-CFE transfer of security labels and the associated source addresses.

Monitoring and Control- The CFE's and CCP's cooperatively capture an extensive log of security events (security range violations, illegal connection attempts, alarm occurrences, etc.). CCP's maintain a comprehensive data base of these system events to provide an audit trail of attempted or inadvertent security violations. For CFE's within its domain, the CCP can request health/status information and can initiate various security and communications tests (alarm tests, loop-back tests, etc.) The CCP operator establishes thresholds for reporting system and security event audit data. The CCP operator can assign configuration parameters (data rates, protocol options, RS-449 options, etc.) and cause them to be downline loaded to its community of CFE's. Fully secure inter-domain communications are managed by CCP-to-CCP coordination. A major responsibility of the CCP is the maintenance

(establishment, updating) and distribution of its CFE's network address/ translation database.

Communications Interfaces- The CFE is normally installed in a network access communications link between a host computer and a packet switch (also known as an Interface Message Processor, or simply IMP). The host is characterized as a data terminal equipment (DTE), the IMP as a data circuit terminating equipment (DCE). The CFE's physical interfaces conform to RS-449 and MIL-STD-188-114A. The link protocol (level 2) is LAPB. The network access protocol (level 3) is the DDN "Standard Service" version of X.25. Over X.25 the CFE will support Internet Protocol (IP, MIL STD 1777). The above "external" protocol suite is implemented on both the Red and Black sides of the CFE. In addition, CANEWARE executes other "internal" protocols to accommodate its own Host -CFE, CFE-CFE, and CFE-CCP transactions. These include end-to-end encryption protocols, status messages, configuration data, etc.

SDNS Relationships- Secure Data Network System (SDNS) is a current NSA sponsored program with the objective of developing and promoting security architecture standards for a wide variety of data communications, particularly packet switched networks (PSN's) and local area networks (LAN's). It is expected that the resulting standards and technology will dominate future secure data network systems and equipment. SDNS compatibility/ interoperability is a program objective. It is a design goal of CANEWARE to comply with the evolving standards of SDNS. If CANEWARE's leading schedule disallows SDNS interoperability for the initial CFE equipment the design will provide for later accommodation via software update. An SDNS related effort is the development of a Key Management Center (KMC). The KMC will be established and operated by NSA to provide FIREFLY keying material for the data world. CANEWARE will use this facility to obtain its FIREFLY key vectors. This will exploit the economies-of-scale achievable by such a "public utility" approach to key management vis-a-vis the establishment of key distribution centers dedicated to individual communities of CANEWARE users.

5.0 PERFORMANCE -

CANEWARE will provide "high performance" security services to eliminate encryption choke points in the network. The basic performance determining parameters for X.25 operations are:

Input/Output Rate	- to 1.544 Mbs
Data Throughput	- >750 Kbs
Packet Rate	-130 Packets/sec
Processing Delay	- <15 ms

Note that the above parameters are for full-duplex operation; the equipment can sustain these rates while simultaneously processing traffic in both directions. Processing

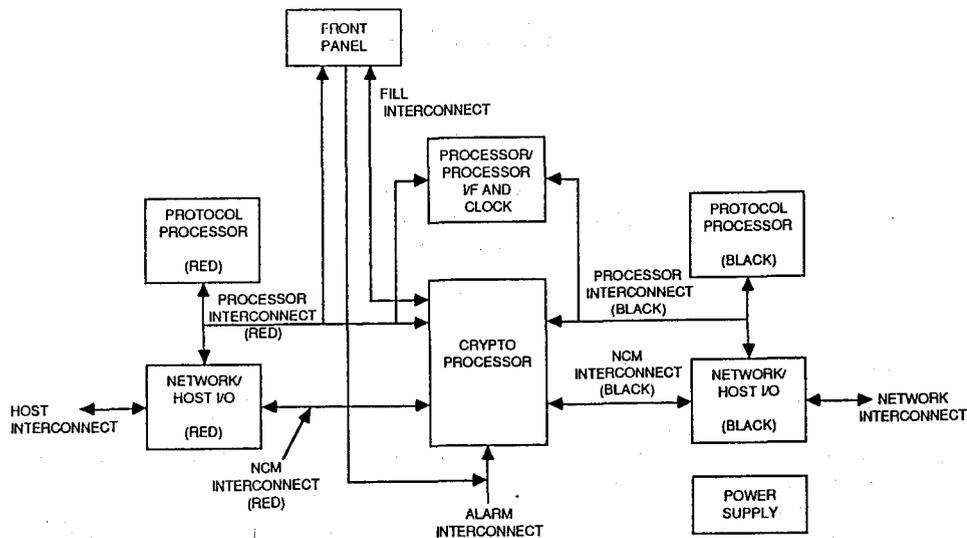


FIG 1. CANEWARE Front End Block Diagram

techniques are designed to enhance performance. For example, permanent network and cryptographic connections are supported for critical and frequent addressees to reduce set-up time.

6.0 IMPLEMENTATION and OPERATION -FIGURE 1 is a block diagram of the CFE hardware. Identical RED and BLACK side Protocol Processors each incorporate an MC68020 32-bit microprocessor and supporting electronics. The Crypto-Processor's main functions are encryption, decryption, key variable storage, FIREFLY operations, and alarms. The Crypto-Processor uses several custom VLSI chips. The Network/Host I/O hardware is also identical on both the RED and BLACK sides (software is different). The principal I/O tasks are data flow-control at the external interfaces and to/from the crypto-processor. These functions are performed by a custom VLSI multi-channel DMA controller, specialized physical and link level I/O chips, and support electronics. The equipment includes a front panel key pad and an 80-character display, to facilitate a menu-driven operator interface. CANEWARE's custom software includes approximately 45K lines of code for the CFE and 20K lines of code for the CCP. The CCP also incorporates commercial operating system and data base management software. Subscriber hosts are "trusted" to properly label all outgoing data with its security classification. This implies hosts have a COMPUSEC rating commensurate with the range of data that they are handling. This label is placed in the Internet Protocol Security Option (IPSO) field of the datagram. The host is expected to provide a reliable transport protocol above IP to assure that host-to-host data is reliably delivered.

7.0 SUMMARY OF CANEWARE FEATURES -The following is a summary listing of the of the principal features of the CANEWARE system:

- *Packet Switched Network Security
- *High Speed Architecture
 - Throughput >750 Kbs, F/D
 - 130 packets/sec, F/D
- *DOD IP/X.25 Protocol Suite
- *Access Control
- *Extensive Built-In Test
- *Configurable I/O, Communications, and Security options
- *Cryptographic Data Protection
- *Multi-Level Security (B2 COMPUSEC)
- *State-of-the-Art FIREFLY Key Management

8.0 PROGRAM STATUS/FUTURE - The CANEWARE program is targeted at 1990 production of operational equipment. The development schedule is:

- CFE
 - Prototype Available -May 1988
 - First E-model CFE -July 1988
- Hardware/Software Integration -August 1988
- CCP
 - Hardware/Software Integration -August 1988
- SYSTEM
 - CFE/CCP Integration -Oct. 1988
 - System Verification -March 1989

FUTURE
 - In the future it is expected that the CANEWARE "product line" will be expanded to include; GATEWAYS, LAN ENCRYPTORS, and TERMINAL ACCESS EQUIPMENT.

Ina Flo: The FDM Flow Tool

Steven T. Eckmann

West Coast Research Center
System Development Group
Unisys Corporation

Abstract

A new information flow tool for the Ina Jo specification language is described. The flow tool is built into the Ina Jo language processor, and generates flow conjectures that are proven with the Interactive Theorem Prover. The flow tool is being used for covert channel analysis in ongoing A1 development projects.

1. Introduction and Overview

The Formal Development Method (FDM) includes the *Ina Jo*TM formal specification language, a processor for the Ina Jo language, and the Interactive Theorem Prover (ITP), for proving theorems generated by the Ina Jo language processor. For more information about FDM see [6], [11], [12], and [13].

Ina Flo, an information flow tool for the Ina Jo specification language, aids covert channel analysis of multilevel secure (MLS) systems. Ina Flo is built into the Ina Jo language processor, and is invoked either with a command-line flag to the Ina Jo processor, or by including a flag in the specification. Ina Flo accepts the entire Ina Jo language, although certain features of the Ina Jo language are not amenable to automated flow analysis. These features are discussed in section 2.1.

Ina Flo actually includes two flow tools. One, henceforth called *MLS*, is similar to those described in [5] and [10], and also has similarities to Mitre's Flow Table Generator [8]. The other implements the Shared Resource Matrix approach [7]. Only the *MLS* tool is discussed in this paper.

We believe Ina Flo is unique among automated flow tools for its scope: it accepts the entire Ina Jo language, including nondeterministic specifications; it accepts arbitrary (lattice-based) security policies, including variable labels; it provides varying levels of support, depending on completeness of the security policy specification.

Information flow in Ina Jo specifications is discussed in section 2. Section 3 presents requirements and guidelines for the use of *MLS*. Section 4 describes a preprocessor for Ina Flo that helps in writing deterministic specifications. The Appendix summarizes the parts of the Ina Jo language used in this paper,

and contains an example demonstrating *MLS*.

2. Information Flow in Ina Jo Specifications

The term "information flow" applied to a state machine model refers to flow of information from one entity in some state to another (possibly the same) entity in a subsequent state. In the Ina Jo language the entities from which information may flow are variables and formal parameters of transforms. The entities to which information may flow are variables. State transitions are modeled by transforms.

An imprecise statement of the definition of information flow used in *MLS* is as follows:

(*) If the new value of y depends on the old value of x then information flows from x to y (written $x \rightarrow y$).

It is assumed that y is a declared variable. Its new value is the value it has after the effect of a transform. It is further assumed that x is either a variable or a transform formal parameter (transform formal parameters are treated as read-only variables). The old value of x is the value it had before the transform. The meaning of the phrase "depends on" is determined by the semantics of the specification language.

The lattice model [2] is built into *MLS*, to the extent that *MLS* assumes (and forces the user to prove) that the security policy included in a specification is a lattice. Therefore, the following rule may be used for determining security:

(**) A flow $x \rightarrow y$ is secure if and only if $MLS_Label(y)$ dominates $MLS_Label(x)$.

In other words, information may securely flow to entities at the same or higher levels, but not to entities at lower (or incomparable) levels. This rule introduces the MLS_Label of a variable, which represents the variable's sensitivity label, and the relation *dominates*, which represents an arbitrary user-specified partial order relation on sensitivity labels. A more rigorous statement of the flow model in Ina Flo is in preparation.

The work reported on here was supported by the NCSC.

TM Ina Jo is a trademark of Unisys Corporation.

2.1. Nondeterminism and Incompleteness

It is not always possible to determine precise dependencies between old and new values of variables, because Ina Jo specifications may be nondeterministic. For example, if the effect of a transform is $N^A = N^B$ then A and B have the same new value, but nothing is known about this value, other than its type. Every state variable might be referenced in deriving this new value. Therefore, it *may be possible* to infer something about the old value of *any* variable from the new value of A (or B), and thus there are potential flows from every variable to A and to B .

Another example of a nondeterministic effect is: $N^X = 1 \mid N^X = 2$. Here the new value of X is certainly 1 or 2, but the specification does not tell us which, nor how the choice is made. Again any state variable may be referenced in deciding whether the new value of X is 1 or 2, so there are potential flows from every variable to X .

In our limited experience with the flow tool to date we have not found it useful to generate flow conjectures for transforms with nondeterministic flows. Therefore, *MLS* produces a false conjecture for transforms that contain nondeterministic flows, along with a list of the nondeterministic flows. We plan to make this behavior optional, since there may be cases in which nondeterministic flows can be proven secure, e.g., when all nondeterministic flows are to System High. In general, the flow tool user will likely find it more useful to defer covert channel analysis to a lower (deterministic) level of specification. Nondeterminism is discussed further in section 4.

Another difficulty in trying to do information flow analysis of Ina Jo specifications is that they need not be functionally complete. That is, an Ina Jo specification need not represent every operation that may be performed by an implementation. See part IV(C) of [1] for a discussion of this point. Ina Flo obviously cannot find flows in operations that are not specified as transforms, so use of Ina Flo on incomplete specifications is not advisable.

3. The MLS Flow Tool

MLS generates conjectures (one for each transform) that, if true, guarantee that no storage channels are in the specified system. Realistically, some flow conjectures will usually not be provable; these represent potential covert channels for which manual analysis will be necessary. Even if all the conjectures are true, there is no assurance that an implementation will not have storage channels, unless the code is formally shown to perform all and only the actions specified as Ina Jo transforms.

3.1. Specifying a Security Policy

Following Denning [2], an information flow policy is defined by a lattice (SC , *dominates*), where SC is a set of security classes, and *dominates* is a binary relation partially ordering the classes of SC . From this definition it follows immediately that *dominates* must be reflexive, transitive and anti-symmetric,

and that the set SC must contain a least upper bound and a greatest lower bound.

To make this definition practical, users must be provided the means to specify security policies. For *MLS* this is done by building into the specification language the capability to specify:

- (1) a set of sensitivity labels (security classes),
- (2) an ordering relation for these labels,
- (3) a label associated with each variable and transform.

The mechanisms for doing these things are described in the following subsections. See [4] for details.

3.1.1. Declaring Labels

Any declared constant or variable may be used as a label, with the restriction that each label must have type `MLS_Label`. This type name is built into *MLS*, but it is not built into the Ina Jo processor. Therefore the user must declare `MLS_Label` explicitly. `MLS_Label` may be any unspecified or specified type. Examples of valid declarations of `MLS_Label` are

```
TYPE MLS_Label
TYPE MLS_Label = (U, C, S, TS)
TYPE Class = (U, C, S, TS),
    Category,
    Categories = Set of Category,
    MLS_Label = Class << Categories.
```

The first example leaves `MLS_Label` completely unspecified; it will presumably be specified more fully either with axioms or at a lower level of specification. The second example declares `MLS_Label` to be an enumerated type with four values. The third example matches the usual notion of security labels in the paper world.

3.1.2. Ordering the Labels

The user must specify the relation *dominates* if *MLS* is to generate flow conjectures; otherwise *MLS* generates lists of flows, as in [8]. *Dominates* may be specified partially or fully with axioms. However, the specification *must* include enough information to prove the following conjectures, which ensure that the specification's security policy is a lattice:

- (1) $A^{\text{lev}}: \text{MLS_Label} (\text{dominates}(\text{SysHi}, \text{lev}))$
- (2) $A^{\text{lev}}: \text{MLS_Label} (\text{dominates}(\text{lev}, \text{SysLo}))$
- (3) $A^{\text{lev}}: \text{MLS_Label} (\text{dominates}(\text{lev}, \text{lev}))$
- (4) $A^{\text{lev1}, \text{lev2}}: \text{MLS_Label} ($
 $\text{dominates}(\text{lev1}, \text{lev2})$
 $\& \text{dominates}(\text{lev2}, \text{lev1})$
 $\rightarrow \text{lev1} = \text{lev2})$
- (5) $A^{\text{lev1}, \text{lev2}, \text{lev3}}: \text{MLS_Label} ($
 $\text{dominates}(\text{lev1}, \text{lev2})$
 $\& \text{dominates}(\text{lev2}, \text{lev3})$
 $\rightarrow \text{dominates}(\text{lev1}, \text{lev3}))$

(1) asserts that `SysHi` is the least upper bound of the set defined by the type `MLS_Label`, and (2) asserts that `SysLo` is the corresponding greatest lower bound. The labels `SysHi` and `SysLo` are built into *MLS*, but are not built into the Ina Jo processor, so they must be explicitly declared as constants (or zero-state definitions) of type `MLS_Label`. (3) asserts that `dominates` is reflexive, (4) that it is anti-symmetric, and (5) that it is transitive. The five conjectures are built into *MLS* as assumptions¹, but they are *not* built into the ITP. Examples follow:

```
TYPE MLS_Label
CONSTANT SysHi, SysLo: MLS_Label,
         dominates(MLS_Label,MLS_Label):boolean
```

This is the minimum that must be specified if you want *MLS* to produce flow conjectures. The type `MLS_Label` is unspecified, as are labels `SysHi` and `SysLo` and the relation `dominates`. The conjectures (1)–(5) will usually not be provable unless they are included as axioms in the specification.

```
TYPE MLS_Label = (U, C, S, TS)
DEFINE SysHi == TS,
        SysLo == U,
        dominates(11,12:MLS_Label) == 11 >= 12
```

`SysHi` and `SysLo` are defined to be the greatest and least element, respectively, of type `MLS_Label`, and `dominates` is defined by the “>=” operator. Since the ITP knows that “>=” is a partial order, conjectures (1)–(5) will be provable from this type declaration and the three definitions.

3.1.3. Associating Labels with Variables and Transforms

To permit the association of security labels with variables and transforms, it was necessary to extend the syntax of the Ina Jo language. In the following example, `SysLo` is a constant, `p` is a formal parameter, and each of the other identifiers is a variable.

```
CLEARANCE Object @ Object_Level,
         Object_Level @ SysLo,
         Buffer(p) @ Proc_Level(p)
```

This declares `Object` to have (variable) security class `Object_Level`, `Object_Level` to have (constant) security class `SysLo`, and `Buffer(p)` to have (variable) security class `Proc_Level(p)` (for all `p` in the proper domain). Each of the expressions on the right side of the ‘@’ must be a *Clearance_Expression*, defined in [4].

3.2. Conjectures Generated

MLS generates a single flow conjecture for each transform in a level. This conjecture is of the form

```
Criteria & Invariants & Effect
->
Secure(Flow1) & ... & Secure(Flown)
```

¹ *MLS* does not presently make use of the transitive or anti-symmetric properties of `dominates`.

If `Flowj` is the flow from `Sourcej` to `Targetj` under condition `Condj`, then `Secure(Flowj)` is defined as

```
Condj -> New_of(MLS_Label(Targetj))
           dominates MLS_Label(Sourcej)
```

The `MLS_Label` function takes a (variable or transform parameter) name and returns a *Clearance_Expression*. The `New_of` function takes a *Clearance_Expression* and returns the same *Clearance_Expression*, with all variable references replaced with “N” variable references. If any `Targetj` or `Sourcej` does not have an associated label, then the conjecture generated for that transform will be `False`, and all the flows for that transform will be listed, as in [8].

In addition to the flow conjectures, *MLS* forces the user to prove the assumptions (1) – (5) listed in section 3.1.2, to ensure that the information built into *MLS* follows from the specification.

4. The *MLS* Preprocessor

The Ina Jo language allows many forms of nondeterminism. We pointed out earlier (section 2.1) that whenever a state transition is nondeterministic, the flow tool makes the conservative (i.e., secure) assumption that an implementation may reference (the image of) every state variable. Therefore, it is important that a specification intended for flow analysis be as deterministic as possible.

We have developed a preprocessor, henceforth called *PREMLS*, which can be used to ensure that some forms of nondeterminism do not appear in the specification seen by the Ina Jo processor. *PREMLS* is invoked via a command-line flag to the `inajo` command, and acts as a filter: it reads an Ina Jo specification and produces a new, presumably more deterministic, specification. In the remainder of this section we discuss *PREMLS*, and present guidelines for writing deterministic specifications.

PREMLS makes deterministic specifications easier to write (and read) by providing short-hand notation for certain expression forms required by the Ina Jo semantics of *No-change*.² The user writes a nondeterministic Ina Jo specification, and *PREMLS* tries to make the specification deterministic by augmenting transform effects with *No-change* clauses, which assert that some variables do *not* change under some conditions. *PREMLS* performs two kinds of *No-change* augmentation often regarded by Ina Jo users as tedious to do manually and unnecessarily difficult to read. This augmentation occurs only in transform effects. The rest of a specification will be unchanged.

Use of *PREMLS* is not required; *PREMLS* is merely a convenience for specification writers unaccustomed to the Ina Jo

² [6] and [9] both include discussions of Ina Jo *No-change* semantics, and explain why the Ina Jo convention is preferable for formal specification to the “no primed occurrence” convention of *SPECIAL*, even though the latter is more convenient.

language. Any specification that could be presented to *MLS* via *PREMLS* could also be presented to *MLS* directly, by writing a deterministic specification in the first place.

4.1. Augmentation of Array Updates

Consider the effect

(1) $N^A(x) = y$

This is a nondeterministic specification of the new value of state variable *A*. The above expression means:

(2) $A^{i:\text{Type-of-}x} (N^A(i) = (i = x \Rightarrow y \langle \rangle N^A(i)))$

In English this says that, in the new state, the x^{th} element of variable *A* has a known value (namely, the old value of *y*), but every other element of *A* has an unknown value. ($N^A(i) = N^A(i)$ is a standard Ina Jo way of indicating an unknown value; all we know about $N^A(i)$ is that it is equal to itself.)

It is often the case that a specification writer wishes to indicate that a finite number of elements of a parameterized variable may change, with all other elements unchanged. One way of doing this is:

(3) $A^{i:\text{Type-of-}x} (N^A(i) = (i = x \Rightarrow y \langle \rangle A(i)))$

The difference between (2) and (3) is that (3) specifies $N^A(i) = A(i)$ for all but the single element that is explicitly changed. Therefore, (3) is entirely deterministic.

PREMLS expands expressions like (1) to the deterministic form exemplified by (3). However, to avoid any confusion about which semantics are expected by the specification writer, *PREMLS* will perform this expansion only if the expression in (1) is enclosed in brackets:

(4) $[N^A(x) = y]$

A specification could thus include both kinds of no-change semantics. Expression (1) will not be expanded, and will be interpreted by the flow tool as (2). Expression (4) will be expanded to the deterministic expression (3).

We call these bracketed expressions *array updates*. Note that the brackets are not part of the Ina Jo language, so a specification that contains array updates must go through the preprocessor before it can be submitted to the Ina Jo processor. The general form of an array update and further examples are presented in [4].

4.2. Augmentation of Conditionals

An Ina Jo transform typically represents some system function that either performs a state changing action, or "returns" an error code. For example, consider the effect

(9) (Exception_1 => N>Error = E1
 <> Exception_2 => N>Error = E2
 <> ...
 <> N"State = Some_expression)

We would like to be able to interpret this as saying that if any error condition occurs, then signal the error, otherwise update

the state. The problem here is again that the semantics of no-change do not agree with this interpretation. Making the Ina Jo meaning of this example explicit, we have

(10) (Exception_1 => N>Error = E1
 & N"State = N"State
 <> Exception_2 => N>Error = E2
 & N"State = N"State
 <> ...
 <> N"State = Some_expression
 & N>Error = N>Error)

from which one can see that every case leaves some part of the new state undefined. This may be the intention of the specification writer, but giving such a specification to Ina Flo would probably be a waste of time, because *MLS* would assume that information flows from every state variable to *State* in the Exception cases, and from every state variable to *Error* in the final case.

The intent of (9) is more commonly

(11) (Exception_1 => N>Error = E1
 & NC"(State)
 <> Exception_2 => N>Error = E2
 & NC"(State)
 <> ...
 <> N"State = Some_expression
 & NC"(Error))

PREMLS augments conditionals as necessary to change expressions like (9) into expressions like (11). In general, *PREMLS* ensures that each branch of a particular conditional modifies the same variables as every other branch of that conditional. This is true for every conditional in every transform effect (but *not* for conditionals in maps or constraints). *PREMLS* does *not* ensure that every branch modifies the same elements of the same variables; that is an unsolvable problem, so *PREMLS* ignores parameters when augmenting conditional expressions. For example, given the expression

(12) (B1 => N"W(a) = 32
 <> B2 => N"X(b) = 33
 <> N"V = 0)

PREMLS would produce

(13) (B1 => N"W(a) = 32 & NC"(X,V)
 <> B2 => N"X(b) = 33 & NC"(W,V)
 <> N"V = 0 & NC"(W,X))

which is still nondeterministic. If the specification were instead

(14) (B1 => [N"W(a) = 32]
 <> B2 => [N"X(b) = 33]
 <> N"V = 0)

then *PREMLS* would produce the deterministic specification

(15) (B1 => A^{i:\text{Type-of-}a} (N"W(i) =
 (i = a => 32 <> W(i)))
 & NC"(X,V)
 <> B2 => A^{i:\text{Type-of-}b} (N"X(i) =
 (i = b => 33 <> X(i)))
 & NC"(W,V)
 <> N"V = 0 & NC"(W,X))

5. Summary

The *Ina Flo* flow tool is currently being used on at least one internal and one external A1 [3] development project. We expect the internal project to suggest improvements in the *SRM* tool; one external project has already suggested numerous improvements in the *MLS* tool, and we expect to continue refining the *MLS* tool to make it more useful for A1 covert channel analysis.

One area where improvements will be made is in handling variable security labels. We believe *MLS* is now secure, but it is too conservative, in that the conjectures it generates for variable labels are often much stronger than necessary to ensure security. The appendix includes an example of this.

Acknowledgements

I am grateful to Deborah Cooper and Paul Eggert for reviewing numerous drafts of this report, and to David Ellis, Morrie Gasser, Sue Landauer and Roger Schell for helpful discussions and comments about the *MLS* part of the flow tool.

References

- [1] Berry, D.M., "Towards a Formal Basis for the Formal Development Method and the Ina Jo Specification Language", *IEEE Transactions on Software Engineering*, 13(2), February 1987.
- [2] Denning, D., "Cryptography and Data Security", pp. 265-278, Addison-Wesley, 1982.
- [3] Department of Defense, "Trusted Computer System Evaluation Criteria", DoD 5200.28-STD, December 1985.
- [4] Eckmann, S., "Ina Flo User's Guide", TM-8416/000/00, Unisys Corporation, June 1987.
- [5] Feiertag, R.J., "A Technique for Proving Specifications are Multilevel Secure", Technical Report CSL109, Computer Science Laboratory, SRI International, January 1980.
- [6] Guttman, J.D., "The Ina Jo Specification Language: A Critical Study", RAD-TR-86-47, Rome Air Development Center and the Mitre Corporation, July 1986.
- [7] Kemmerer, R., "Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels", *ACM Transactions on Computer Systems*, 1(3), August 1983, Pages 256-277.
- [8] Kramer, S., "The MITRE Flow Table Generator — Volume I", M83-31 Volume 1, The MITRE Corporation, January 1983.
- [9] Platek, R. and D. Sutherland, "The Semantics of the Feiertag MLS Information Flow Tool and its Impact on

Design Verification: Some SCOMP Examples", Odyssey Research Associates, 1984.

- [10] Rushby, J., "The Security Model of Enhanced HDM", Computer Science Laboratory, SRI International, August 1984.
- [11] Scheid, J., S. Anderson, R. Martin, S. Holtsberg, "The Ina Jo Specification Language Reference Manual — Release 1", TM 6021/001/02, System Development Corporation, January 1986.
- [12] Scheid, J. and S. Holtsberg, "Ina Jo Definition", TM 7527/016/00, System Development Corporation, March 1986.
- [13] Smith, G. and D.V. Schorre, "The Interactive Theorem Prover (ITP) User's Manual", TM 6889/000/06, System Development Corporation, December 1986.

Appendix

The following table contains a summary of the less obvious Ina Jo syntax used in the paper.

the notation...	means...
A"	for all
N"	new value of
NC" (v)	N"v = v
->	implies
(b=>s<>t)	if b then s else t

The following example demonstrates input to and output from the Ina Jo processor when the *MLS* option is selected. The example specification (shown in figure 1) is written for *PREMLS* — note the '[' and ']' brackets in some of the transforms. Also note that the *INHIBIT* flag is used to suppress the correctness and consistency conjectures ordinarily produced by the Ina Jo processor.

Figure 1 is a specification of a simple resource manager that uses the low water mark security policy.

Figure 1 — Example specification *lwm.ina*

```

$TITLE Low Water Mark example for Ina Flo
SPECIFICATION Low_Water_Mark
LEVEL Top_Level MLS Inhibit

TYPE
  t, /* this is the object type */
  Process,
  MLS_Label

CONSTANT
  dominates(MLS_Label, MLS_Label): Boolean,
  SysLo, SysHi: MLS_Label,
  Proc_Level(Process): MLS_Label

```

```

VARIABLE
  Object, Buffer(Process): t,
  Curr_Proc: Process,
  Object_Level: MLS_Label

CLEARANCE
  Object      @ Object_Level,
  Buffer(p)    @ Proc_Level(p),
  Curr_Proc   @ SysLo,
  Object_Level @ SysLo

AXIOM
  A"lev: MLS_Label ( Dominates(lev,lev) )
  & A"lev: MLS_Label ( Dominates(SysHi,lev) )
  & A"lev: MLS_Label ( Dominates(lev,SysLo) )
  & A"11,12:MLS_Label ( Dominates(11,12)
                        & Dominates(12,11)
                        -> 11 = 12 )
  & A"11,12,13:MLS_Label ( Dominates(11,12)
                           & Dominates(12,13)
                           -> Dominates(11,13) )

INITIAL
  A"p: Process ( Dominates(Object_Level,
                          Proc_Level(p) ) )

CRITERION True

TRANSFORM Read
  EFFECT
    ( Dominates(Proc_Level(Curr_Proc),
                Object_Level)
      => [N"Buffer(Curr_Proc) = Object]
      <> NC"(Buffer) )

TRANSFORM Write
  EFFECT
    ( Dominates(Object_Level,
                Proc_Level(Curr_Proc))
      => N"Object_Level =
          Proc_Level(Curr_Proc)
          & N"Object = Buffer(Curr_Proc)
      <> NC"(Object, Object_Level) )

TRANSFORM Reset
  EFFECT
    ( Dominates(Proc_Level(Curr_Proc),
                Object_Level)
      => N"Object_Level = SysHi
      <> NC"(Object_Level) )

CLEARANCE
  Read @ Proc_Level(Curr_Proc),
  Write @ Proc_Level(Curr_Proc),
  Reset @ Proc_Level(Curr_Proc)

END Top_Level
END Low_Water_Mark

```

The features of the specification in Figure 1 that we wish to point out are

- (1) The flag *MLS* appears on the Level line. This causes the Ina Jo processor to invoke *MLS* on level Top_Level.
- (2) There are declarations for *MLS_Label*, *Dominates*, *SysHi* and *SysLo*. Each of these names is built into *MLS*, but not into the Ina Jo processor itself, so they must be declared in the specification if they are to be used.
- (3) The label assigned to *Buffer* is a variable, *Proc_Level*. The implications of this are discussed after figure 2.
- (4) The three assumptions built into *MLS* are explicitly specified. If they were not, the corresponding conjectures generated by *MLS* would (probably) not be provable. This is a consistency check between *MLS* and the specification.
- (5) The clearance declarations for the three transforms are superfluous, because none of the transforms has parameters. In general, a transform requires a clearance specification only if it has parameters, because it is through these parameters that information may flow from the transform invoker to modified variables.

The next figure contains an abridged listing produced by the command `inajo -p lwm`.³ The listing is unmodified, except as noted.

Figure 2 – Listing of `lwm.ina`

```

2-SPECIFICATION Low_Water_Mark
3-LEVEL Top_Level MLS Inhibit
4-
. . . . .
many lines deleted
. . . . .

56-
57-TRANSFORM Write
58-
59-Effect
60- ( Dominates(Object_Level,
61-                Proc_Level(Curr_Proc))
62-   => N" Object_Level =
63-       Proc_Level(Curr_Proc)
64-       & N" Object = Buffer(Curr_Proc)
65-
66-   <> NC"(Object, Object_Level)
67- )
68-

```

³ The '-p' flag causes the Ina Jo processor to invoke the preprocessor on `lwm.ina`, then use the output of the preprocessor as its input.

```

. . . . .
many lines deleted
. . . . .
THEOREM FOR SysHi:
A" lev:MLS_Label(Dominates(SysHi, lev))

THEOREM FOR SysLo:
A" lev:MLS_Label(Dominates(lev, SysLo))

THEOREM FOR Dominates - reflexive:
A" lev:MLS_Label(Dominates(lev, lev))

THEOREM FOR Dominates - Antisymmetric:
A" lev1, lev2:MLS_Label(
    Dominates(lev1, lev2)
    & Dominates(lev2, lev1)
-> lev1 = lev2)

THEOREM FOR Dominates - Transitive:
A" lev1, lev2, lev3:MLS_Label(
    Dominates(lev1, lev2)
    & Dominates(lev2, lev3)
-> Dominates(lev1, lev3)

```

Flow conjecture for Transform Read deleted

```

Flow Conjecture for Transform Write
C&I True
E & ( Dominates(Object_Level
        , Proc_Level(Curr_Proc))
    => N" Object_Level
        = Proc_Level(Curr_Proc)
    & N" Object
        = Buffer(Curr_Proc)
    <> N" Object = Object
    & N" Object_Level =
        Object_Level)
    & N" Curr_Proc = Curr_Proc
    & A" #0:Process(
        N" Buffer(#0) = Buffer(#0))
F1 -> ( Dominates(Object_Level
        , Proc_Level(Curr_Proc))
    & True
    -> Dominates(N" Object_Level
        , Object_Level))
F2 & ( Dominates(Object_Level
        , Proc_Level(Curr_Proc))
    & True
    -> Dominates(N" Object_Level

```

```

Proc_Level(Curr_Proc))
Flow Conjecture for Transform Reset
True
& ( Dominates(Proc_Level(Curr_Proc)
    , Object_Level)
    => N" Object_Level = SysHi
    <> N" Object_Level = Object_Level)
& N" Curr_Proc = Curr_Proc
& A" #0:Process(
    N" Buffer(#0) = Buffer(#0))
& N" Object = Object
-> ( Dominates(Proc_Level(Curr_Proc)
    , Object_Level)
    & True
    -> Dominates(N" Object_Level
    , Object_Level))

```

85-END Low_Water_Mark

The first five conjectures (called THEOREMS), for SysHi, SysLo and Dominates, will be included in the listing and itp file whenever *MLS* is invoked. In this case each of them is proved either automatically by the ITP, or with a single instantiation command by the user. This is expected, since the conjectures are stated in the specification as axioms.

The conjecture for transform Read is also proved automatically, and is not shown in the Figure. The conjecture for transform Reset is not proved automatically, but is easy to prove with the ITP. Transform Write is troublesome, because it downgrades Object (by changing Object_Level). Recall the general form of a flow conjecture:

```

Criteria & Invariants & Effect
->
Secure(Flow1) & ... & Secure(Flown)

```

In the flow conjecture for transform Write in Figure 2, the first line, marked **C&I**, is the conjunction of the criterion and the (implicitly true) invariant. Beginning on the second line, and marked **E**, is the augmented effect of transform Write. Beginning on the line marked **F1** is the security condition for the first potential flow, and beginning on the line marked **F2** is that for the second potential flow.⁴

MLS ensures that no downgrades are allowed by requiring proof that the new value of the (variable) label of the potentially downgraded variable dominates the old value of that label. This requirement is the source of **F1**. Unfortunately, we cannot prove

```

dominates(N"Object_Level, Object_Level)
(unless Object_Level = Proc_Level(Curr_Proc)).
We can argue informally that the transform is secure, because

```

⁴ The markings associated with the flow conjecture for transform Write were not generated by the flow tool; they were added for this paper.

the information in Object in the old state, at level Object_Level, has been replaced in the new state with information at level Proc_Level (Curr_Proc) (in the old state), which is $N^{\text{Object_Level}}$.

This example points out that, although it is possible to use variables as labels, *MLS* is overly conservative about them, in the sense that some secure flows will not be provably secure from the conjectures generated by *MLS*. We are working on this problem.

A GYPSY VERIFIER'S ASSISTANT

Ben L. Di Vito and Larry A. Johnson

TRW Defense Systems Group
One Space Park
Redondo Beach, CA 90278

Abstract

Current generation tools and techniques for formal verification have inherent limitations that prevent them from being applied on a larger scale. We describe an IR&D effort underway at TRW to augment the Gypsy Verification Environment (GVE) with a knowledge-based "verifier's assistant." The resulting methods and tools will support the construction of deductive theories to extend the practical range of today's formal verification tools. A prototype Deductive Theory Manager (DTM) is being developed to maintain appropriate knowledge bases and interact semi-automatically with the GVE. Candidate knowledge bases are simultaneously under development.

Introduction

Formal verification is the primary distinguishing feature of Division A requirements in the Trusted Computer System Evaluation Criteria [1]. Obviously, the National Computer Security Center (NCSC) attaches considerable importance to formal methods and the A1 level of assurance. A1 currently represents the highest degree of trust recognized by NCSC for multilevel modes of operation.

Substantial progress has been made in applying formal methods to computer security over the last fifteen years. A fair amount of success has been achieved in developing secure operating systems and verifying them to the A1 level. To developers of large scale, mission-oriented systems, however, information security technology in general, and formal verification technology in particular, is still lacking in important areas. For systems designed to meet the C2/B1 level of the Criteria, it can be argued that sufficient technology exists for designing cost effective systems. Nevertheless, it is clear that such an argument cannot be made for systems designed

to operate in multilevel mode, that is, requiring a Trusted Computing Base (TCB) certified in the B2-A1 range.

Formal specification and verification, whether to meet computer security or any other requirements, is one of the most challenging problems facing defense system developers. Whereas verification of operating system kernels has received widespread attention, comparatively little work has gone into other aspects of trusted system verification. Formal verification of trusted applications software, for instance, is largely an unexplored area. It differs considerably from operating system verification efforts where the goal is usually to prove that some well defined security model properties hold. In contrast, verification of applications software involves proving that derived security properties hold. These properties tend to be complex statements of functional behavior and involve much more effort to synthesize and prove than, for example, an information flow property. Current generation verification methods and tools are ill-equipped to cope with the volume and complexity of proofs that are likely to result from a large mass of trusted applications software.

The Gypsy Verification Environment (GVE) [2] is the most commonly used of the NCSC-endorsed formal verification tools for computer or network security. A prime attraction of GVE and the Gypsy language are their applicability to a broad range of tasks:

- Formal statement of security models
- Representation of formal top-level specifications for TCBs
- Formal description of system designs (i.e., a program design language)
- Proof of the preservation of a secure state in security models

- Proof that an FTLS complies with security model requirements
- Proof that the high-level language implementation code satisfies an FTLS

We consider GVE to be the best available methodology of its kind. Nevertheless, in spite of GVE's strengths, the practice of formal verification remains a very demanding engineering discipline. Limitations of even the best available tools and techniques render their application to real system designs arduous. Chief among the current GVE limitations is that proofs require too much user interaction and direction, especially for proofs of concurrent systems. As a result, considerable GVE experience and theorem proving knowledge is required to carry out proofs effectively. Such weaknesses tend to become magnified by the scale factor; as problem complexity grows, using the GVE prover successfully becomes much more difficult.

TRW is addressing these problems as part of our Multilevel Applications Security Technology (MAST) IR&D project. A major portion of this project is devoted to the problems of formal verification for systems of nontrivial size. The remainder of the paper introduces our overall approach to solving these problems. This work is still in its preliminary stages and we anticipate its continuation through 1988.

Objective

Our goal is to advance formal verification technology to better support large scale verification efforts. We have devised a technique to enable verifiers to build *deductive theories* for particular domains of interest. The objective is to be able to manage efficiently the complexity of large scale proofs through knowledge-based augmentations of existing verification systems, GVE in particular. Our ultimate goal is to be able to support the design proof (A1 level of assurance) required for 100K lines of trusted applications software.

More specifically, our objective is to develop a tool that can be used in conjunction with GVE to push the practical limits of formal verification technology. The tool and its associated knowledge bases will support model verification required at the B levels of assurance, and formal demonstration of the correspondence of the formal top-level specification to the model at the A1 level.

Achieving efficient proofs requires structuring them into subproofs and handling each one independently. *Lemmas* are the mechanism to achieve this structuring; their use is an application of the classical divide-and-conquer technique for problem solving. Lemmas are already supported by the GVE in a rudimentary fashion. In addition, other types of user-directed theorem proving operations are provided by the GVE to control proof complexity, including expansion of function definitions, equality substitutions, and instantiation of variables.

While use of these basic features within GVE is simple and straight forward, it is overly burdensome for a user to keep track of all definitions and lemmas, and to know when to make use of them during a proof. Having additional automated support for GVE proofs would greatly extend the range of formal verification technology. It would permit more realistic secure system formal top level specifications and would improve the practicality of code level proofs.

With the capability described herein, applications verifiers could easily build up bodies of deductive knowledge specific to their particular domains of discourse. This would be of value during both the specification and verification phases. Domains relevant to TRW's A1-level specification and verification work are being investigated for their applicability to this approach.

To summarize our overall objective for this effort, we list the following major goals of our proposed tools and techniques:

- Extend basic formal verification technology
- Enable verification of large amounts of trusted software
- Promote reusable verification concepts and results
- Make verification technology more accessible to less sophisticated users

Accomplishing these goals will lead to a significant advance in the technology for developing A1 systems.

Approach Overview

Our approach is based on the introduction of an automated tool that can be thought of as a *verifier's assistant*. Its purpose is to augment the theorem proving capabilities of the GVE with problem-oriented proof heuristics. We refer to this tool as a

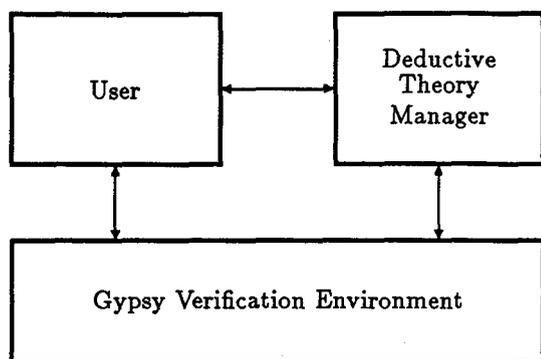


Figure 1: The verifier's assistant.

Deductive Theory Manager (DTM). Figure 1 depicts the high level architecture of the composite verification environment that results.

In this architecture, the DTM plays the conceptual role of a smart user that draws upon a copious body of theorem proving knowledge. This knowledge and associated heuristics are used to supplement the user's own knowledge of the problem and skills at proving theorems. It is important to emphasize that the DTM does not replace the user; the user is still ultimately responsible for directing and understanding the proof process. What we expect, however, is that the combined team of user, DTM, and GVE will be much more effective at verification than a user and GVE alone. The DTM is designed to operate at an intermediate level of detail with respect to verification problem solving. Thus, the GVE continues to be concerned with low level details as the "verification engine," but now we are able to raise the user to a higher level of abstraction, freeing him to worry about more global issues in the overall verification problem.

The DTM itself relies on knowledge-based techniques for its implementation. In particular, we are basing our effort on the Knowledge Engineering Environment (KEE) tools developed by IntelliCorp [3]. KEE is a commercially available software product for implementing expert systems and knowledge-based components. We will use KEE to realize our DTM concept and maintain the various knowledge bases needed to support formal verification activities.

In Figure 1, we show three separate interfaces between the various entities. The user-GVE interface is the same as it normally is, except for a slight extension of the user commands to support user-to-DTM requests. In the normal "on-line" use of our

configuration, the user carries out proofs via commands to the GVE. When requested to get help from the DTM, the GVE will interact with it via the GVE-DTM interface. The user only communicates directly with the DTM in an "off-line" mode for the purpose of building and maintaining the knowledge bases.

To illustrate the type of interaction proposed for the user-DTM-GVE team, consider the following scenario for conducting a proof.

- Suppose a user is trying to prove a theorem of the form

$$H_1 \wedge \dots \wedge H_n \Rightarrow C.$$

- Assume a knowledge base has been built (previously proven lemmas and heuristics for GVE theorem proving) and that information about the theorem has been supplied by GVE (current goal, types, function definitions, lemmas, etc.).
- The user issues a command to the GVE to request assistance from the DTM (through a *Deduce* command). A typical DTM response would be to return a set of actions resulting from matching a particular set of rule conditions in the knowledge bases.
- The GVE performs the indicated actions by manipulating the H_i or C as appropriate.

The actions that can be prescribed by the DTM include all the inference rules normally available to an ordinary user through prover commands.

The work to be completed in 1987 is as follows:

- Development of the theoretical methods required to support a deductive theory manager.
- Implementation of an interface from the GVE to the DTM package using KEE to allow them to execute cooperatively on the Symbolics Lisp Machines.
- Development of several preliminary knowledge bases.
- Construction of a prototype of the DTM software.
- Evaluation of the effectiveness of the DTM on proofs developed on other TRW projects. Specifically, the number of interactive user steps will be compared with and without the DTM.

Methodology

The following sections summarize the essential concepts of our composite verification methodology.

Deductive Theories

Formal verification is an application of mathematical logic. A logician's concept of *theory* is a set of valid formulas, that is, formulas that are either axioms or can be proved from the axioms and previously proved theorems. In the jargon of ordinary mathematics, a theory would include axioms, definitions, and theorems. Our concept of deductive theory includes the conventional logical concept of theory, in the context of the Gypsy language, but also extends it by including heuristics for GVE theorem proving. Thus, a deductive theory contains axioms, theorems, and meta-information for proving new theorems.

The importance of deductive theories is that by gradually developing a theory and storing it in a knowledge-base, an increasingly more powerful set of facts is available for constructing new proofs in the future. The effort to prove complex theorems is greatly reduced in the presence of a rich body of previously proven theorems. In the absence of such knowledge, proofs must be derived from first principles, which will undoubtedly require much more work to complete. A well organized theory, on the other hand, allows for reuse of previous effort and provides the obvious economies. Good [4] has claimed that the development of reusable theories is a vital part of making formal verification practical for realistic applications.

Our approach recognizes the importance of building, maintaining, and using deductive theories in the verification process. We are pursuing a knowledge-based implementation to capture deductive theories and organize the knowledge in maximally useful ways. Emphasis will be placed on organizations for efficient search and retrieval of relevant information at suitable points in a proof. We see the use of deductive theories as one of the few practical, near-term ways to make verification technology "scale up" to the level of realistic system sizes.

It is useful to think of deductive theories as being organized into natural hierarchies. We see the need to support four distinct layers of theories and their corresponding knowledge bases.

1. The lowest level theory contains general knowledge relevant to virtually all GVE proofs. Typically this would involve properties of the pre-

defined Gypsy types and operators (those properties not already provided by GVE itself).

2. Domain specific theories include special information related to broad classes of applications (e.g., operating systems, database management).
3. The next layer is project specific. It contains information related to the particular approach and architecture of a single application.
4. A fourth theory is strictly personal. It enables a user to define rules that are helpful to him, but may not be useful to another person's style of specification and proof.

Verification Strategy

The introduction of deductive theories and their automated support permits new ways of organizing effort on large projects. In particular, deductive theories allow for a very effective division of labor based on the relative verification skills of project members. We can draw an analogy between verification and software development. It is customary to divide software development efforts into two major types: systems software and applications software. The systems programmers build a base for the applications programmers to utilize. Likewise, verification effort could be divided into two types: the development of common-use deductive theories by one group, and the verification of applications by another, which makes use of the theories developed by the first group.

This division of labor allows us to take maximum advantage of those with the better verification skills. In general, it takes more skill to "design and implement" a deductive theory than it does to make use of one to prove properties of an application. Therefore, by dedicating the more experienced verification talent to theory development efforts, we can maximize the utilization of scarce human resources.

Proof Tactics

The GVE theorem prover accommodates several different tactics for carrying out proofs. There are approximately 20 major inference rules that can be invoked by a user; the DTM will likewise be able to invoke these same inference rules. It is up to the theory developer and knowledge-base designer to define the KEE rules so that these commands will be invoked at the appropriate times.

Two of the GVE commands and associated proof tactics are especially important and worthy of mention. Assume we are trying to prove a theorem of the form

$$H_1 \wedge \dots \wedge H_n \Rightarrow C.$$

The *use* and *claim* inference rules together with other information allow the prover's attention to be directed to very specific goals.

Use

This command takes an instance of a Gypsy lemma L and adds it to the current goal as an extra hypothesis:

$$L \wedge H_1 \wedge \dots \wedge H_n \Rightarrow C$$

The DTM would *use* a lemma drawn from one of its knowledge bases. The lemma would be a previously proven fact that had been duly recorded in the GVE's database. Typically, the prover would be directed to *Proceed* after this point to continue with the GVE's own proof heuristics. Alternatively, additional DTM-supplied commands might form the proof continuation.

Claim

This command introduces a new boolean-valued expression Q as a formula that is claimed to follow from the hypotheses. Two new cases must be proved as a result:

$$\begin{aligned} H_1 \wedge \dots \wedge H_n &\Rightarrow Q \\ H_1 \wedge \dots \wedge H_n \wedge Q &\Rightarrow C \end{aligned}$$

The first shows that the claim Q is valid and the second that Q can be used to prove the original conclusion. A useful special case occurs when Q has the form $P \Rightarrow C$ and P follows automatically from the hypotheses. This makes the proof of the original conclusion from the claim trivial and leaves the real work in trying to establish the validity of the claim. It is thus a convenient way for the DTM to derive lemmas "on-the-fly" when a suitable one does not already exist. Note that this technique constitutes a form of *backward chaining*, although *claim* could be used to achieve *forward chaining* as well. Similarly, the *use* command could be used to achieve either forward or backward chaining.

Specification Style

In order to make the DTM approach more tractable, we should refrain from allowing just any syntactically and semantically correct piece of Gypsy specification to be used. Instead, theory developers should

```
function secure_state
  (s: protection_state): boolean =
begin
  exit (assume result iff
    simple_security_property(s)
    & star_property(s)
    & discretionary_security_property(s));
end;
```

Figure 2: Sample security model definition.

adopt and prescribe a style for writing specifications that facilitates subsequent analysis by the DTM using their theories. The specific style chosen is not nearly as important as the fact that one is adopted. This permits the DTM to make simplifying assumptions concerning the form of expressions, bodies of function definitions, etc. Coordinating the specification writing conventions with the proof strategy will yield significantly better results in the long run.

Examples of styles we are adopting are illustrated in Figures 2 and 3. These are based on a Gypsy rendition of the Bell-LaPadula security model [5]. Key features of the *get.read* specification style are its state parameter, exception condition parameter, conditional exit assertion, and encapsulation of conditions in the function *valid.get.read*. Figure 4 abstracts the general form of this specification style. If the DTM and its knowledge bases can assume a similar structure in all associated operations, they can make use of more efficient rules tailored to the general style.

Verification Condition Schemas

In the same interest of tractability, we will also make the DTM cognizant of the forms that verification conditions (VCs) will take. For many computer security applications, such as proving that operations are security preserving, the VCs will occur in a small number of special forms. For example, the case of proving that an operation is security preserving has the following general form:

$$\text{secure}(S_1) \wedge \text{effects}(S_1, S_2) \Rightarrow \text{secure}(S_2)$$

Figure 5 shows this form in an actual VC. By insisting that specifications for each operation be written in the same way, the VCs for all the operations will have the same form. This greatly facilitates the work of knowledge-base designers.

```

procedure get_read
  (s: subject; o: object;
   var ps: protection_state;
   var dec: rule_decision) =
begin
  exit if valid_get_read(s, o, ps')
  then dec = granted
  & ps = ps' with
    (.b := ps'.b <:
     access(s, o, read))
  else dec = denied & ps = ps'
  fi;
end;

function valid_get_read
  (s: subject; o: object;
   ps: protection_state): boolean =
begin
  exit (assume result iff
        (privileged(s,
                     discretionary_exemption)
         or ps.m[s, o, read])
        & dominates(ps.fs[s].sec,
                    ps.fo[o].sec)
        & (privileged(s,
                     security_star_exemption)
         or dominates(ps.fc[s].sec,
                    ps.fo[o].sec)));
end;

```

Figure 3: Sample Gypsy specification.

```

procedure P ( . . . ;
             var s: state;
             var e: exception) =
begin
  exit if valid_P( . . . )
  then e = OK
  & s = s' with ( . . . )
  else e = error & s = s'
  fi;
end;

function valid_P ( . . . ;
                  s: state): boolean =
begin
  exit (assume result iff
        <boolean expression> );
end;

```

Figure 4: General specification form.

```

Verification condition RULE_MACHINE_B#4
H1: RULE (NS) = R1
H2: SECURE_STATE (PS)
H3:  VALID_GET_READ (SUBJ (NS),
                    OBJ (NS),
                    PS)
    -> PS with (
        .B := PS.B
        @ [seq: ACCESS (SUBJ (NS),
                       OBJ (NS),
                       READ)])
    = PS#1 & D#1 = GRANTED
H4:  not VALID_GET_READ (SUBJ (NS),
                        OBJ (NS),
                        PS)
    -> D#1 = DENIED & PS = PS#1
-->
C1: SECURE_STATE (PS#1)

```

Figure 5: Sample verification condition (VC).

Generics

Some of the key advantages of the DTM are that it will enable the user to develop more generic theorems or rules than possible within GVE. For example, in GVE a lemma about some property of sequences must state the specific type of element in the sequence. If the theorem is to hold for fifteen types of elements, fifteen lemmas are required. In DTM, a single rule using a generic type will provide an expression of the concept in terms of all possible element types. When used in a specific proof, the DTM can instantiate a generic lemma to provide GVE with the specific lemma name for the appropriate element type.

Similar generic capabilities are being investigated for functions as well as types. Such capabilities would enable the expression of lemmas in which functions referenced within the lemmas are parameters that get instantiated with actual function names when invoked. This would allow expressions of, for example, transitivity for sets of functions rather than requiring individual rules or lemmas for each case. There are significant theoretical issues, however, to be resolved before this concept can be employed. This is an area undergoing further study.

Relation to Automated Theorem Proving

Part of the motivation for introducing the DTM concept is to provide an automated lemma search capa-

bility. There are other mechanical theorem proving systems that provide such features. The Boyer-Moore theorem prover [6], for example, has a very useful facility for storing and retrieving previously proven lemmas. Lemmas in this system have the form of conditional rewrite rules:

$$C_1 \wedge \dots \wedge C_n \Rightarrow LHS = RHS$$

Automatic application of the rules proceeds by first attempting to unify the left-hand-side (LHS) with a term in the formula. After successful unification, the conditions (C_1, \dots, C_n) are established in backward chaining fashion to determine whether the rewrite should take place. There is no way to be any more selective than this in the application of lemmas.

The AFFIRM system [7] also has a very elegant method of automatically applying rewrite rules to a formula. In AFFIRM, however, rewrite rules are unconditional. Hence, there is even less control over the application of rules. This has significant limitations when trying to solve more general problems than operations on abstract data types, which is AFFIRM's primary domain.

Both of these examples represent systems that do an admirable job of supporting reusable deductive theories. However, a good deal more control over the application of lemmas is required when dealing with deep and complex proofs. The widespread coverage attempted by term rewriting systems is effective when proofs are shallow, but they tend to become overwhelmed when attempting proofs requiring deeper penetration.

In this area, we feel the DTM approach offers an advantage for complex proofs. Using DTM, it is possible to define very selective conditions on lemma invocation and thereby channel proofs using highly directed forms of heuristic knowledge. Rather than attempting broad coverage through rewriting, for which GVE would be inappropriate anyway, the DTM concept allows proof guidance that is much more context sensitive. This will lead to fewer lemma invocations, at the expense of more computation to determine what to apply. Nevertheless, we feel this is the correct tradeoff to be making with a system of the sort that GVE represents. It allows more of the verification analyst's skill to be encoded into the knowledge bases and relies less on brute force coverage principles.

Prototype Implementation

TRW is currently engaged in developing a prototype of the DTM concept to assess and demonstrate its feasibility. Following is a description of the implementation features.

DTM Design

TRW's DTM prototype is hosted on Symbolics Lisp Machines. The DTM acts as an advisor and helper for the user doing proofs. Figure 6 shows the relationship of the user to both the GVE and the DTM. We expect that there will be multiple knowledge bases used simultaneously. In fact, the prototype supports the simultaneous use of four knowledge bases in keeping with the previous description of four layers of deductive theories.

TRW's DTM design enables the user to perform proofs in any manner he desires with or without its help. If a user wishes the help of the DTM, he makes his request through the following added GVE prover commands:

- **Advise.** The user is provided with a list of recommended steps to be requested of GVE.
- **Deduce.** The DTM determines the same set of recommended steps as with the Advise option and proceeds to feed the requests automatically to the GVE.

In the *Advise* option, no actual steps are taken in the proof of the verification condition or lemma. In the *Deduce* option, however, the proof steps are performed. It may turn out that the user does not want the particular steps taken by the DTM. In this case, the user still has all of the standard GVE recovery options available, including the capability to back up (erase) one or more proof steps.

For the prototype DTM, TRW is combining GVE with one of the commercially available expert system tools: KEE by IntelliCorp. KEE will enable us to establish a prototype within a short period of time without spending time building inference engines and knowledge maintenance capabilities. Furthermore, KEE offers sophisticated front-end graphics to provide a very efficient user interface to the DTM knowledge bases.

For this application, the GVE has to be extended only to request the DTM to provide *advice* or *deduction*. The information provided by GVE to the DTM is only theorem status information (e.g., current theorem, data types, function definitions, and

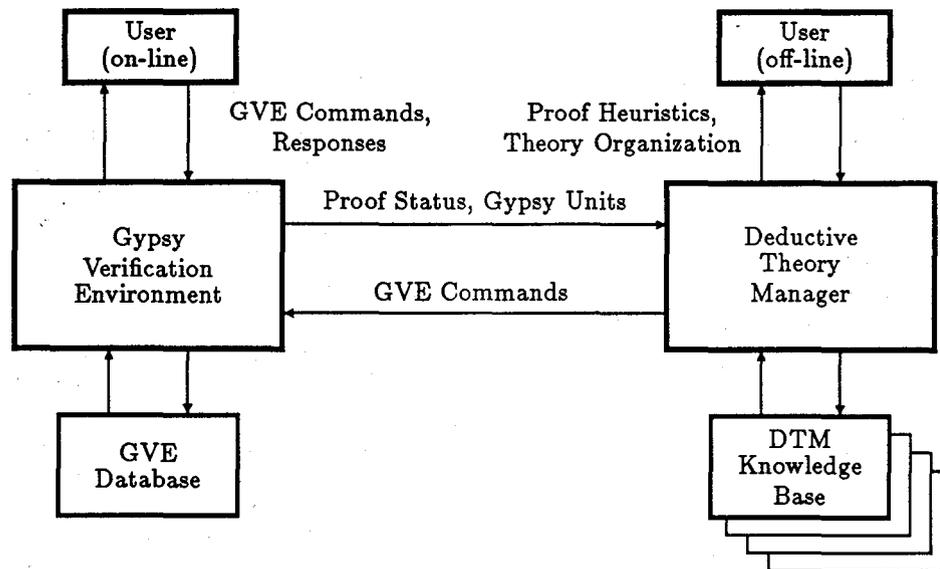


Figure 6: Deductive Theory Manager Architecture.

lemmas). The DTM only reads this information; it does not directly modify it or add to it. A set of Lisp interface functions has been introduced to retrieve information from GVE's internal database for use by the KEE rules.

Similarity Function

One of the central features required to make the DTM work is the ability to determine the similarity between expressions. A *similarity function* has been defined which computes the similarity between two predicates. It can be used, for example, to compare a hypothesis to the conclusion or to compare the consequent of an implication type of hypothesis with the conclusion.

The function works on expressions represented in Gypsy internal format; all symbolic operations are represented in prefix format. The function returns the number of similarities and the number of differences. All differences are returned in the form of pairs indicating the symbolic differences. By returning the actual differences, the differences can be further reduced by utilizing additional information contained in other hypotheses or by use of instantiations if Gypsy Skolem variables are present.

Knowledge Base Design

A KEE knowledge base contains a set of rules that constitute rules-of-thumb or heuristics for guiding the GVE theorem proving process. The actions of a rule form a set of GVE user level commands to be performed. A few of the simpler rules we envision for the DTM are shown in Figure 7 using the syntax of KEE. The actions are either displayed for the user as advice or sent automatically to the GVE, depending on the user's command choice.

The first rule in the figure states that if there is an equality in one of the hypotheses, it will be used to make a substitution. For example, consider the following theorem:

$$P(x, f(y)) \wedge y = z \Rightarrow P(x, f(z))$$

By making the substitution of z for y , it is clear that the first hypothesis will unify with the conclusion to complete the proof. A more selective version of this rule having additional conditions for triggering the substitutions would be more desirable in practice.

The second rule states that if the conclusion of the theorem refers to a function that is not mentioned in any of the hypotheses, the definition of the function is necessary to continue the proof. In GVE terms this generally means the function must be expanded. The case of nonexpandable Gypsy functions will also be considered.

The third rule provides a simple mechanism for

```

(IF (the hypothesis of DTM is ?Hyp)
  (Lisp (Equality ?Hyp))
  (?H-Num = (Get-Num ?Hyp))
  THEN DO
  (GVE EqSub ?H-Num))

(IF (the conclusion of DTM is ?Cncl)
  (?Hyps = (Get-Hyps))
  (?Func = (Find-Func ?Cncl))
  (Lisp (Not-In ?Func ?Hyps))
  (Lisp (Find-Func ?Func))
  THEN DO
  (GVE Expand ?Func))

(IF (?Hyps = (Get-Hyps))
  (the conclusion of DTM is ?Cncl)
  (the lemma of DTM is ?Lemma)
  (?LCncl = (Get-Cncl ?Lemma))
  (?LHyps = (Get-Hyps ?Lemma))
  (?Diff = (Similar ?Cncl ?LCncl))
  (Lisp (Make-Equal ?Cncl ?LCncl))
  THEN DO
  (GVE Use ?Lemma))

(IF (the conclusion of DTM is ?Cncl)
  (Lisp (Buf-Sequence ?Cncl))
  (?Hyps = (Get-Hyps))
  (?OBuf = (GOBuf ?Cncl))
  (?IBuf = (GIBuf ?Cncl))
  (?TOBuf = (GBuf ?IBuf ?Hyps))
  (?TIBuf = (GBuf ?OBuf ?Hyps))
  THEN DO
  (GVE Claim
   ?TOBuf sub ?IBuf and
   ?OBuf sub ?TIBuf
   -> ?OBuf sub ?IBuf))

```

Figure 7: Example theorem proving rules.

applying lemmas that have been defined in the Gypsy application. Even though the lemmas are within the Gypsy database, the GVE theorem prover is not aware of their existence until the theorem prover user identifies the applicability of a lemma. The user introduces a lemma into the current proof by the command *use lemma-name*. DTM rules can help automate the process of identifying applicable lemmas as is shown by the third example rule. The use of the SIMILAR function during a search will locate a lemma with a conclusion similar to the current goal. Once found, the MAKE-EQUAL function at-

tempts to make the two expressions equal by making appropriate instantiations of any Skolem variables.

The fourth rule is much more complex. It addresses a common type of problem encountered in proving properties of concurrent systems. If, for example, one wishes to show that security is maintained as a message flows through a system consisting of multiple processes, it must be shown that security is maintained as it flows through each of the individual processes. This fourth rule accommodates such a proof by setting up the steps for a transitivity argument.

The DTM rules will be designed to complement GVE's capabilities. GVE already has some built-in heuristics, the effects of which can be seen when the user issues the *Proceed* or *QED* prover commands. The DTM will synthesize higher level proof heuristics by combining sequences of ordinary GVE prover commands.

Tool Endorsement Issues

One of the key features of the TRW approach is the use of a technique that will provide the functionality of proof heuristics while, at the same time, not affect the soundness of proofs. The DTM is only useful if the final proofs are performed with tools endorsed by the NCSC. Recall that the DTM interface to GVE is logically equivalent to that of a smart user. Consequently, the only information fed to the GVE are standard GVE theorem proving commands that the real user could have typed if he were able to think of them. There is no direct modification of the proof tree, or any other internal GVE data structures, by the DTM. Therefore, the GVE remains totally responsible for ensuring soundness, just as it does when an ordinary user is entering commands.

Due to this partitioning of GVE and DTM environments, we claim that the DTM should not be subject to NCSC endorsement. The NCSC or any other certification authority could validate proofs performed under a GVE-DTM configuration quite easily. All that is required is to capture the DTM-generated prover commands and merge them with the user-generated commands. Then a replay of the proofs on a conventional GVE system that has not been outfitted with a DTM should suffice to establish the validity of the proofs. Our DTM design will contain the features necessary to support such a validation activity.

A further advantage of our DTM approach with respect to GVE integrity is that its knowledge base is extensible by the user. Thus, automating new

proof heuristics does not require modifications to the GVE itself, with all the attendant ramifications on NCSC tool endorsement. One need only add the appropriate rules to the DTM knowledge bases.

Development Status

The GVE and KEE systems have been integrated under the Symbolics 6.1 operating system version. The GVE uses Zetalisp and the Lisp functions called directly by the DTM rules are in Common Lisp (supported by KEE). The design places GVE in control with the DTM as a supporting function. The user invokes DTM to provide advice or automatic proof support. The only required change to GVE was to extend the GVE theorem prover grammar to support the two new commands. The DTM function calls provide a copy of the current goal by copying the expression from a local GVE variable to a global variable accessible by KEE. Several functions had to be written to extract the type definitions, function definitions, and lemmas contained in the GVE database.

The major effort in the DTM implementation is found in the development of a set of primitive functions that facilitate manipulation of the GVE symbolic expressions. KEE is best suited for reasoning about objects which either have no internal structure or are readily represented as structures of fixed composition (e.g., an object has weight, size, color). While KEE poses some difficulties in the manipulation of symbolic expressions, it offers some advantages in terms of tracing the reasoning process. It has a full set of tools that allow the tracing of both the forward and backward chaining process.

Most of the early work has been in the development of rules that can be applied whenever the necessary preconditions of the rules are satisfied. The focus of the future work is on extending DTM to be more of a planning process. Our objective is to have DTM determine a series of GVE theorem prover steps that will complete the proof for a theorem. The DTM will proceed forward as long as it can make progress. It will backup whenever progress is not possible and consider alternative strategies. One of the difficulties is that the DTM must be able to compute the effect of the application of each of the Gypsy theorem prover commands. For commands like BACKCHAIN, it is fairly simple. For commands like SIMPLIFY, it is much more difficult. One of the possibilities being explored is to use the GVE Simplifier.

Conclusion

TRW has developed a novel concept to enhance formal verification technology. We expect that the full elaboration of the DTM approach will lead to a substantial advance in verification capability. The primary benefit of this work is an increase in the effective range of applicability of Gypsy-based verification efforts. This should make feasible A1 development efforts that currently are considered impractical due to the potentially large amount of trusted software required.

Our plans are to continue developing and evaluating our DTM prototype during 1987. An official GVE-to-DTM interface is planned by employing the services of Computational Logic, Inc. In 1988, we expect to refine our DTM design based on what is learned in 1987. In addition, we will focus on the serious development of knowledge bases for various problem domains and assess the gains realized through the addition of a DTM capability.

References

1. National Computer Security Center, "Department of Defense Trusted Computer System Evaluation Criteria," DoD 5200.28-STD, December 1985.
2. D. I. Good, B. L. DiVito, and M. K. Smith, "Using the Gypsy Methodology," Institute for Computing Science, University of Texas at Austin, June 1984.
3. IntelliCorp, "KEE, Software Development System User's Manual," Version 3.0, 1986.
4. D. I. Good, "Reusable Problem Domain Theories," Technical Report 31, Institute for Computing Science, University of Texas at Austin, September 1982.
5. D. E. Bell and L. J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," Technical Report ESD-TR-75-306, Mitre Corporation, Bedford, Mass., March 1976.
6. R. S. Boyer and J. S. Moore, *A Computational Logic*, Academic Press, New York, 1979.
7. S. L. Gerhart, et al, "An Overview of AFFIRM: A Specification and Verification System," *Proceedings IFIP 80*, October 1980.

Formal Models, Bell and LaPadula, and Gypsy

Tad Taylor
Bret Hartman

Research Triangle Institute
P.O. Box 12194
RTP, NC 27709

ABSTRACT

An approach for developing formal security models is presented. It is accompanied by a technique for expressing and proving models in Gypsy. The approach is adapted and generalized from the Bell and LaPadula model, as presented in *Secure Computer System: Unified Exposition and Multics Interpretation*.

1. INTRODUCTION

The *Trusted Computer System Evaluation Criteria* [Criteria] requires class B2 or higher systems to have "A formal model of the security policy supported by the TCB..." There has been uncertainty concerning how efforts to meet this requirement could be meaningfully and productively incorporated into a system development effort, and little has been said about the practical application of existing modeling concepts. Furthermore, minimal guidance exists for integrating existing formal specification and verification technology with the task of developing these formal models. It is our hope that this paper will shed some light on these issues.

For those who would rather apply existing formal verification technology than develop it, a technique for expressing and proving models in Gypsy and an approach for developing a general class of formal models is presented. The approach is derived and adapted from [Bell76] and involves expressing a system's functionality and desired security properties in a state machine format. By adapting the existing Bell and LaPadula format, we inherit the concept of a secure system being a sequence of secure states (with important caveats to be mentioned later), the basic security theorem, and the concept of rules of operation (but not the particular rules described in [Bell76]).

Bell and LaPadula has been subjected to the "social process" within the formal verification community for many years. Although this scrutiny has identified shortcomings (e.g. - see [McLean87]), much of the basic framework of the model is still attractive. By adapting Bell and LaPadula, our approach has the important advantage of being expressed in terms already familiar to our audience.

Experience has also shown that Bell and LaPadula is not appropriate for all formal security modeling efforts. Likewise, we are not claiming that our adaptation is universal or is the final word in formal modeling. However, we have attempted to generalize our approach so that a larger class of systems can be modeled.

Section 2 of this paper describes the goals for our approach. Section 3 presents a high level view of the modeling techniques. Finally, Section 4 gives a detailed example of a Gypsy model in the context of a message filter application. The discussions assume that the reader has a general familiarity with the Bell and LaPadula model as described in [Bell76] and, to a lesser extent,

with the Gypsy language [Good84].

2. MODEL FEATURES

To fully appreciate this approach, it is necessary to understand what we hope to accomplish by developing a model and what benefits we think the process can provide. The model should not be thought of as a more abstract FTLS (Formal Top Level Specification). The model is more than that and we feel there is an advantage in treating the development of the model separately from the development of the FTLS. Formal modeling can be an effective method for developing the formalisms that will be used to prove the FTLS.

The clear and accurate presentation of system properties is a model's most important objective. Unfortunately, no approach can directly aid the model developer in this process. However, our approach has many of the details worked out for the basic framework and allows a group to concentrate on the issues related to their specific problem.

The modeling process should accomplish more than this basic requirement. A point often overlooked is that the process of developing and writing a model should provide insight into how security requirements integrate with system functionality. Understanding this relationship is imperative. This is especially useful (if not critical) when the requirements and functionality have been defined by a group separate from the system developers.

In the "best of all possible worlds", models are written in the initial stages of a system's development, when design or architecture decisions are not yet made or are very, very tentative. To avoid changing and reworking the model to accommodate fluid designs, the model should be isolated from the architecture and specifics of the implementation. This practice can represent a substantial reduction in modeling effort. A second reason for isolating the model is to avoid placing unnecessary constraints on the eventual implementation. The notion of "unnecessary constraints" is somewhat nebulous, but in general, the model should deal with requirements, not design. A state machine approach satisfies this need quite naturally.

3. A TECHNIQUE AND APPROACH FOR MODELING

This section presents a high level description of an approach for developing formal models. Because of the approach's generality and the variety of situations for which it is applicable, it is not possible to present a detailed series of steps to tell the developer how to build a formal model. An example is more appropriate, since it can give further insight into our approach. Section 4 presents an explanation of how this technique is applied to develop a model for a simple message filter.

An explicit goal of the approach is to maintain a strong correlation with the Bell and LaPadula model, as expressed in [Bell76]. We feel that our ideas and techniques are faithful enough to these concepts to develop a model **exactly** as it appears in [Bell76] and adaptable enough to apply the concepts of the model to a wider range of situations, such as having different security properties, modeling systems other than general purpose operating systems, or capturing more complete notions of security.

The approach, in the spirit of [Bell76], consists of three steps:

- [1] Defining a "framework"; a framework is analogous to the descriptive capability and general mechanisms referred to in [Bell76].
- [2] Defining system functionality which is analogous to the specific solutions facet.
- [3] Proving system security, which is the process of proving that the functionality specified in [2] obeys the constraints defined in [1].

3.1. Defining a Framework

By "framework," we mean the foundation upon which the model is based. A framework provides an arena for describing a system of a particular class but does not delve into the intended functionality of the system being modeled. For example, in [Bell76], the framework represents a security kernel for a general purpose operating system. The specific system that this is applied to is Multics.

Two issues addressed in the framework are the mechanics of the model itself and tailoring the approach to a certain class of systems. Bell and LaPadula referred to these two issues as their model's descriptive capability and general mechanisms, by which they attempt to capture the basic elements of a computer system and the limitations to be placed on such a system's effects to maintain the defined notion of security. "Mechanics of the model" refers to the state machine architecture and the style in which a system is defined to be secure (not the actual security properties). Tailoring the system deals with the selection of security properties for a particular class of system application and identifying the class of system to be modeled.

3.1.1. Mechanics of the Model

The mechanics of the model describe how the state machine view operates and the mechanisms used to reason about the state machine. If a state machine model is desired, our approach takes care of several fundamental issues. The approach addresses issues such as how to represent the state machine, how to represent the security requirements, how to relate them to the state machine, and how to express the state transitions. We are

not claiming that using the approach allows these issues to be ignored completely. They must be addressed, but only in terms of how they need to be adapted for a specific application - a much easier topic to deal with than starting from scratch.

We present details of how the mechanics are established in section 4, but in short, the notions of subject, object, and security level track rather straightforwardly from [Bell76]. An "action" is a 4-tuple of (R,D,"new state," "old state") and the term "wset" is the action set, W, from Bell and LaPadula. The representation in Gypsy of security level and its associated concepts of classification, category, etc. is also very similar.

3.1.2. Tailoring the Framework

The largest differences in defining the framework between this approach and [Bell76] are the notions of state and security properties. A class of system is defined by the state components, as these are the only things upon which rules can act. In [Bell76], a state is a 4-tuple of (b,H,M,f), representing the current access set, the hierarchy, the access permission matrix, and the security function, respectively. This notion of state allows rules to deal with the functionality of a security kernel. However, this is too limited for many security applications. To expand or redefine the potential functionality of the system, it is necessary to change the state components of [Bell76]. Our approach supports this by not having any prescribed set of state components, allowing whatever is relevant for a particular system.

As for security properties, many people consider the sole significance of the Bell and LaPadula model to be the "simple security property" and "*-property". We definitely do **not** feel this is the case. The model is much more complex than this belief indicates, establishing a view of system interaction with its environment. The concepts of actions, rules, and the definition of what makes a system secure are fundamental to the model. The simple security and *-properties are only used to define the secure state predicate. We allow a different security predicate to be defined, while maintaining the same conceptual framework. In this manner, our approach can be tailored to applications where some variant of the simple security and *-properties, or perhaps a completely different definition of secure state, is more appropriate.

However, defining security strictly as a state invariant is inadequate to capture our intuitive notion of security. It is necessary to place restrictions on state transitions as well. To provide this capability, we augment the model's concept of security with a set of rule properties - conditions that must be met by all rules. The "tranquility" property is a typical example of such a condition.

3.2. Defining System Functionality

When the state is defined, the limits of what a system may do are defined, but nothing is said about what the system will actually do. A framework only provides a potential for certain forms of action; it does not guarantee what will happen. For example, the framework defined in Section 4 provides a capability to read in, process, and send out messages, but it does not guarantee that anything will happen. The framework only requires that whatever does happen obeys the security requirements set forth.

The functionality of a specific system is defined by a set of rules of operation. Rules of operation are state transitions

describing under what conditions changes can be made to the state. The rules must completely capture all possible effects on the state of which the system is capable.

3.3. Proving System Security

Once the actual operation of the system is defined by the rules of operation, it is necessary to ensure that the system is secure. In our approach, two conditions must be satisfied before a system can be said to have met its requirements. The first condition is that the system defined must meet the constraints defined in the predicate "secure_system" and the second is that the system specific rules of operation meet all of the rule security properties. Both are vital to capturing a complete notion of security. Satisfaction of the first condition is shown by proving a theorem stating that if a rule set is composed of the defined rules, and if the action set is defined from those rules, and if the initial state is valid, then the system defined by the action set must be secure. The second condition is met if every rule defined has the specified property.

3.4. Caveat

This approach has been presented in a very linear fashion. This will not be the case in practice. There is iteration among all three steps until a complete model is produced. Then, if either the system's requirements or functionality is redefined during the development process, the model should be modified to reflect those changes and reproved.

4. AN EXAMPLE OF THE APPROACH IN GYPSY

The idea of representing abstract model concepts in Gypsy is still new to many people's minds. They are used to thinking of Gypsy as dealing in more concrete terms, as is the case in performing code verification. However, the Gypsy specification language, being a form of typed first order logic, is sufficiently powerful to represent modeling concepts. This section describes the techniques used in our approach in the context of a Gypsy example.

The example we use is a simple message filter. Messages arrive at the filter on various incoming lines, and if they pass the security checks, the messages get routed to outgoing lines.

We chose a message filter as an example for our modeling approach because it is very different from the traditional use of Bell & LaPadula as a secure operating system model. The example illustrates that state machine modeling in the Bell & LaPadula framework is applicable to a wide array of security applications and not simply secure operating systems.

The formal model for the message filter consists of four Gypsy scopes. The `FILTER_Global_Definitions` scope defines the types and functions used as a basis for the framework. Each of the remaining three scopes corresponds to a step of our approach, as described in Section 3: the framework is defined in the `FILTER_Security_Policy`; the system functionality is defined in the `FILTER_Rules_of_Operation` scope; and the steps to prove that the system functionality obeys the constraints of the framework are defined in the `FILTER_Rules_Model_Proof` scope.

The remainder of this section contains the Gypsy formal specifications for each of these scopes. English text is interspersed with the Gypsy to help the reader understand its general

organization as well as the more subtle specification techniques. Name importations between scopes have been omitted from the Gypsy for the sake of brevity. Other than the omission of name importations, the sections contain a complete listing of the Gypsy scopes, which have been parsed and proved using the Gypsy Verification Environment.

4.1. Defining the Framework

The definition of the framework for the FILTER model begins in the `FILTER_Global_Definitions` scope. This scope contains definitions of the primitive components. The `FILTER_Security_Policy` completes the framework by defining the descriptive capability and the general mechanisms of the model.

4.1.1. Primitive Components

This scope begins by defining all of the necessary model primitives such as subjects, objects, and the state. Similar elements of Bell & LaPadula appear as comments on the right margin.

```
scope FILTER_Global_Definitions =
begin
{ Elements of the FILTER }      { B & L similar elements }
For the model of a message filter, subjects are considered to
be lines which receive messages into the filter and send mes-
sages out. The objects are the messages. Each message
contains its security_level, sender, receiver, and
contents.
type subject = line;                { S }
  type line = pending;
type object = message;              { O }
  type message = record(s1: security_level;
                        sender: line;
                        dest: line;
                        contents: contents);
    type contents = pending;
Security Attributes are defined as in Bell & LaPadula.
type classification = pending;      { C }
type category_set = set of category; { K }
  type category = pending;
type security_level =                { L }
  record (classification: classification;
         category_set: category_set);
function dominates (s1, s2: security_level) : boolean =
begin exit (assume result =
  ( s2.classification le s1.classification
    & s2.category_set sub s1.category_set));
end;
```

Request and **decision** are used as in Bell & LaPadula to indicate a rule invocation request and the corresponding indication of success or failure. The security levels of subjects and objects are defined as in Bell & LaPadula with one exception: the security level of a given subject has the potential to change, while the security level of a given object cannot. This view is consistent with message processing, where changing the security level of a message changes the message itself, and creates a distinctly new message.

```

type request = pending;           { R }
type decision = (yes, no, audit); { D }
type subject_security_level =    { Fs }
  mapping from subject to security_level;
function object_sl(o: object): security_level = { Fc }
  begin exit (assume result = o.sl);
  end;

```

Gypsy buffer histories are used to describe the incoming and outgoing objects that each subject can access. A buffer history maintains a copy of all traffic over a given buffer. The information is represented as a sequence of objects, the order of the sequence representing the order of processing. The buffer histories replace the (subject, object, access) triple of Bell & LaPadula. Histories are a more appropriate means of describing the security properties of a message application than access triples, which work better in the secure operating system domain. While we have chosen to use the concept of buffer histories in the model, we do not wish to use the Gypsy IPC (interprocess communication) mechanism. The complexity of the Gypsy send and receive mechanisms are unnecessary for the model.

```

{Buffers}                               { B }
type buffer_histories = mapping from subject to
  buffer_history;

```

```

  type buffer_history = sequence of object;

```

A state is a record structure of the state variables. It includes the buffer histories associated with the incoming and outgoing lines, as well as the security levels of the subjects. In order to keep the security policy independent of the concrete representation of the state, several extraction functions are defined to retrieve items from the state.

```

type FILTER_state =                    { V }
  record(incoming: buffer_histories;
         outgoing: buffer_histories;
         subject_sl: subject_security_level);

```

```

{ State Extraction Functions }

```

```

function incoming_history(s: subject; v: filter_state):
  buffer_history =
  begin exit (assume result = v.incoming[s]);
  end;

```

```

function outgoing_history(s: subject; v: filter_state):
  buffer_history =
  begin exit (assume result = v.outgoing[s]);
  end;

```

```

function subject_sl(s: subject; v: filter_state):
  security_level =
  begin exit (assume result = v.subject_sl[s]);
  end;

```

Actions characterize the effects that may take place in the system. An action consists of four components: a request, a decision, a starting state and a resulting state. A collection of actions is called an **Action Set** and is intended to include all combinations of requests, decisions, new states, and old states. The term **wset** in the security policy scope refers to an action set analogous to "W" in Bell and LaPadula.

```

type action_set = set of action;       { W }
type action = pending;
function action_request(a: action): request = pending;
function action_decision(a: action): decision =
  pending;
function action_old_state(a: action): filter_state =
  pending;
function action_new_state(a: action): filter_state =
  pending;

```

Rules, as defined by Bell and LaPadula, are a function from a (request, state) pair to a (decision, state) pair. Since properties must be proved about sets of rules, it is difficult to represent rules as Gypsy functions, because Gypsy does not provide a capability to reason about sets of arbitrary functions. Instead, we define rules as a Gypsy mapping from a **rule_input** to a **rule_output**. Rule inputs represent a request and an input state. Rule outputs represent a decision and an output (resultant) state. In Gypsy, this means that when a rule is supplied a **rule_input** it returns a **rule_output**.

```

type rule = mapping from rule_input to rule_output;
  { rho in B&L }

```

```

type rule_input = record (request: request;
  state: filter_state);
function input_request(ri: rule_input): request =
  begin exit (assume result = ri.request);
  end;
function input_state(ri: rule_input): filter_state =
  begin exit (assume result = ri.state);
  end;

```

```

type rule_output = record (decision: decision;
  state: filter_state);
function output_decision(ro: rule_output): decision =
  begin exit (assume result = ro.decision);
  end;
function output_state(ro: rule_output): filter_state =
  begin exit (assume result = ro.state);
  end;

```

```

end; { scope FILTER_Global_Definitions }

```

4.1.2. Definitions and Relationships

The role of the security policy scope is to finish the task of establishing the model's framework, begun by the global definitions scope. This is done by defining basic system concepts, such as what it means for a system to be secure, and defining a view of how components work together.

```

scope FILTER_Security_Policy =

```

```

  begin

```

```

    { name declarations omitted }
  end;

```

In [Bell76], a system's operation is defined by its action set, **W**. The action set is a **complete** representation of all possible actions for the system. Similarly, the concept of a system, in our model, is captured by an action set, called **wset**. In Bell and LaPadula, a system is secure iff (if and only if) all possible state sequences are secure. A state sequence is secure iff every state in the sequence is a secure state. Thus, a system is secure iff every reachable state is secure. Our definition of **secure_system** captures the same concept by stating that for any state that is a **valid_state**, that state must be secure.

```

function secure_system(wset: action_set;
                      z0: filter_state): boolean =
begin exit (assume result iff
  all v: filter_state,
  valid_state(v, wset, z0)
  ->
  secure_state(v));
end;

```

Secure_State defines what it means for a state to be secure. In this case, for all objects and subjects, if an object is in the outgoing history of some subject in the state, then the properties expressed by **security_checks_made** must hold. These properties are that the object must be in some subject's incoming buffer history and that the security level of the object must dominate the security level of the receiving subject. Furthermore, the security level of the exporting subject must dominate that of the object.

```

function secure_state(v: filter_state):
                                boolean =
begin exit (assume result iff
  all obj: object,
  all sending_subj: subject,
  obj in outgoing_history(sending_subj, v)
  ->
  security_checks_made(obj, sending_subj, v));
end;

function security_checks_made(obj: object;
                              sending_subj: subject;
                              v: filter_state): boolean =
begin exit (assume result iff
  some receiving_subj: subject,
  obj in incoming_history(receiving_subj, v)
  & dominates(subject_sl(sending_subj, v),
              object_sl(obj))
  & dominates(object_sl(obj),
              subject_sl(receiving_subj, v)));
end;

```

Valid_State defines the valid states of the system. A valid state is any state which is either the initial state, or the new state in some action in the action set.

```

function valid_state(n: filter_state;
                    wset: action_set;
                    z0: filter_state):
                                boolean =
begin exit (assume result iff
  (n = z0
  or
  (some a: action,
   a in wset
   & n = action_new_state(a)));
end;

```

The **basic_security_theorem** states that a system is secure iff the action_set is secure and the initial state is secure. This demonstrates that the system can be proved secure by proving that all actions are secure.

```

lemma basic_security_theorem (wset: action_set;
                              z0: filter_state) =
  secure_system(wset, z0)
  iff
  ( secure_state(z0)
  & secure_action_set(wset));

function secure_action_set (wset: action_set):
                                boolean =
begin exit (assume result iff
  all a: action,
  a in wset
  ->
  secure_state(action_new_state(a)));
end;

```

Rule security properties are a mechanism for filling in the gaps that a secure state invariant cannot address. These properties are expressed by a pair of functions. The first function defines a constraint on a rule set, namely that each rule in the set possess some property. The second function defines this property for an individual rule. In the case of **FILTER**, there are three rule security properties.

Secure_state_preserving_rule_set is used to prove **secure_system**. A rule set is security preserving if and only if every rule is security preserving. A rule is security preserving when, for every input/output pair, the output state is secure whenever the input state is secure.

```

type rule_set = set of rule;

function secure_state_preserving_rule_set(rs: rule_set):
                                boolean =
begin exit (assume result iff
  all r: rule,
  r in rs
  ->
  secure_state_preserving_rule(r));
end;

function secure_state_preserving_rule (r: rule):
                                boolean =
begin exit (assume result iff
  all ri: rule_input,
  all ro: rule_output,
  ( r[ri] = ro
  & secure_state(input_state(ri))
  ->
  secure_state(output_state(ro)));
end;

```

The tranquility property is a classic example of the problems with relying completely on state invariants to preserve a desired concept of security. Without this a requirement, the security levels of subjects can be manipulated to "satisfy" the secure state requirements, without conforming to the intended behavior. In this system, a rule set is tranquility preserving iff no rules change a subject's security level.

```

function tranquility_preserving_rule_set (rs: rule_set):
                                boolean =
begin exit (assume result iff
  all r: rule,
  r in rs
  ->
  tranquility_preserving_rule(r));
end;

function tranquility_preserving_rule (r: rule):
                                boolean =
begin exit (assume result iff
  all ri: rule_input,
  all ro: rule_output,
  all subj: subject,
  r[ri] = ro
  ->
  subject_sl(subj, input_state(ri)) =
  subject_sl(subj, output_state(ro)));
end;

```

A rule set is buffer history preserving if and only if none of the rules are able to remove objects from any buffer histories. This is necessary to ensure that the buffer histories we have defined conform to the semantics of Gypsy buffer histories. Without such a requirement, a rule could "satisfy" the conditions of secure state by simply rewriting the buffer history.

```

function buffer_history_preserving_rule_set (rs:
                                     rule_set): boolean =
begin exit (assume result iff
  all r: rule,
  r in rs
  ->
  buffer_history_preserving_rule(r));
end;

function buffer_history_preserving_rule (r: rule):
                                     boolean =
begin exit (assume result iff
  all ri: rule_input,
  all ro: rule_output,
  all subj: subject,
  all o: object,
  r[ri] = ro
  ->
  incoming_history(subj, input_state(ri))
  =
  incoming_history(subj, output_state(ro))
  & (o in outgoing_history(subj, input_state(ri))
  ->
  o in outgoing_history(subj, output_state(ro))));
end;

```

The theorem `secure_rules_form_secure_system` corresponds to the Bell and LaPadula corollary A2. This demonstrates that a system can be proved secure by proving that all its corresponding rules are secure. It serves as a link between the concept of defining a system in terms of an action set and defining the functionality of a particular system in terms of rules of operation.

```

lemma secure_rules_form_secure_system (rs: rule_set;
                                     wset: action_set;
                                     z0: filter_state) =
( action_set_derived_from_rules(rs, wset, z0)
  & secure_state_preserving_rule_set(rs)
  & secure_state(z0)
  ->
  secure_system(wset, z0);

```

An `action_sequence` is a sequence of actions produced by consecutive applications of rules. This type is not directly required in the model, but serves to provide a link between an `action_set` and a `rule_set`. A particular `action_sequence` describes one path through the action set, chosen by the selection of rules and requests, and resulting in a sequence of (decision, new state) pairs.

```
type action_sequence = sequence of action;
```

An `action_set` is derived from a `rule_set` iff every action in the `action_set` is in an action sequence which starts with the initial state and has been derived from the rule set.

```

function action_set_derived_from_rules (rs: rule_set;
                                     wset: action_set;
                                     z0: filter_state):
                                     boolean =
begin exit (assume result iff
  all a: action,
  a in wset
  iff
  some aseq: action_sequence,
  a in aseq
  & aseq ne null(action_sequence)
  & action_old_state(first(aseq)) = z0
  & action_seq_derived_from_rules(rs,
  nonlast(aseq),
  last(aseq)));
end;

```

An `action_sequence` is derived from a `rule_set` iff the last action is derived from the rule set, its old state is the same as the new state of the previous action, and the rest of the `action_sequence` is

derived from the rule set. This recursive definition links each action in the sequence together.

```

function action_seq_derived_from_rules (rs: rule_set;
                                     aseq: action_sequence;
                                     a: action): boolean =
begin exit (assume result iff
  ( action_derived_from_rules(rs, a)
  & (aseq ne null(action_sequence)
  ->
  ( action_new_state(last(aseq)) =
  action_old_state(a)
  & action_seq_derived_from_rules(rs,
  nonlast(aseq),
  last(aseq))))));
end;

```

An action is derived from a `rule_set` iff there is a rule which, when supplied a rule input corresponding to the action's request and old state, returns a rule output that corresponds with the action's decision and new state.

```

function action_derived_from_rules (rs: rule_set;
                                     a: action):
                                     boolean =
begin exit (assume result iff
  some r: rule,
  some ri: rule_input,
  some ro: rule_output,
  r in rs
  & r[ri] = ro
  & input_state(ri) = action_old_state(a)
  & input_request(ri) = action_request(a)
  & output_decision(ro) = action_decision(a)
  & output_state(ro) = action_new_state(a));
end;

```

This lemma is quite similar to `secure_rules_form_secure_system`, but deals with action sequences instead of action sets. This lemma serves to aid the proof of the other lemma. This is an example of proof modularity in Gypsy.

```

lemma secure_rules_form_secure_action_seq
(rs: rule_set;
 aseq: action_sequence;
 z0: filter_state) =
( aseq ne null(action_sequence)
  & action_old_state(first(aseq)) = z0
  & action_seq_derived_from_rules(rs,
  nonlast(aseq),
  last(aseq))
  & secure_state_preserving_rule_set(rs)
  & secure_state(z0)
  ->
  secure_action_sequence(aseq);

```

An `action_seq` is secure iff the new_state of every action is secure.

```

function secure_action_sequence (aseq:
                                     action_sequence):
                                     boolean =
begin exit (assume result iff
  all a: action,
  a in aseq
  ->
  secure_state(action_new_state(a)));
end;

```

The remaining lemmas are to prove that the definition of `dominates` in the model satisfies the partial ordering requirements of being reflexive, antisymmetric, and transitive.

```

lemma dominates_is_reflexive (s: security_level) =
  dominates (s, s);

lemma dominates_is_antisymmetric (s1,s2: security_level) =
  ( dominates(s1, s2)
    & dominates(s2, s1) )
  ->
  s1 = s2;

lemma dominates_is_transitive (s1,
                               s2, s3: security_level) =
  ( dominates(s1, s2)
    & dominates(s2, s3) )
  ->
  dominates(s1, s3);

End; {Filter_Security_Policy}

```

4.2. System Functionality

The Rules of Operation scope defines the functionality of the system by stating properties of a valid initial state and enumerating all state transitions. These properties must then be proved to be secure with respect to the security policy described in section 4.3. This example makes some simplifications to minimize the size of the specification. First, there is only one rule of operation, called `process_message`. For other applications, our approach can be generalized easily to a larger number of rules. Second, the specification assumes that the incoming buffer history is pre-established at the initial state, and remains fixed for the duration of the execution of the filter. This assumption allows us to avoid creating a rule to describe how a message arrives at the filter. Despite the assumption, the specification is still completely general, since we prove the security of the filter for any arbitrary incoming buffer history.

```

scope FILTER_Rules_of_Operation =
begin
{ name declarations omitted }

A valid initial state for the message filter is defined by
initial_filter_state. A state ZO is a valid initial filter
state iff no messages have been sent out on any outgoing lines,
and the internal mappings are defined for all lines. No restrictions
are placed on the incoming buffer histories at initialization.

function initial_filter_state(z0: filter_state): boolean =
begin exit (assume result iff
  all outgoing_line: line,
  all l: line,
    z0.outgoing[outgoing_line] = null(buffer_history)
  & l in domain(z0.incoming)
  & l in domain(z0.outgoing)
  & l in domain(z0.subject_sl));
end;

```

In this simple example, we have only a single rule of operation. A complex system would have many more, all written in a similar style. `Process_message_rule` is defined as a constant (a function with no arguments) of type `rule`. The rule is defined by comparing it with a corresponding specification function called `process_message`. The definition of the rule says that the mapping returned by `process_message_rule` matches `process_message` for every input/output pair. This approach allows us to make the connection between an arbitrary set of rules, as the policy uses, and an instantiation of a particular set of rules for the system being specified.

```

function process_message_rule: rule =
begin exit (assume
  all ri: rule_input,
  all ro: rule_output,
  result[ri] = ro
  iff
  process_message(ri) = ro);
end;

```

The function `process_message` describes the functionality of the system at an abstract level. The function describes the `rule_output (filter_state, decision)` that it will return given any `rule_input (filter_state, request)`. The function must be well-defined to prevent unsoundness. `Process_message` says that if there is a message which is an element of some incoming line's buffer history, if the security level of some outgoing line dominates the the security level of the message, and if the security level of the message dominates the security level of the incoming line, then a new `filter_state` is created by adding the message onto the outgoing line's buffer history of the old `filter_state`, and the decision is `yes`. If any of these checks are not met, then the decision is `no` and the resulting `filter_state` is unchanged. This specification makes no attempt to describe how the next message to process is determined, or how the message is routed to a proper outgoing line. In this specification, `request` is never used. It is assumed that the rule is activated upon arrival of a message.

```

function process_message (ri: rule_input): rule_output =
begin
  exit (assume
    some m: message,
    some old_state: filter_state,
    some incoming_line, outgoing_line: line,
    old_state = ri.state
    & if m in old_state.incoming[incoming_line]
      & dominates(old_state.subject_sl[outgoing_line],
                  m.sl)
      & dominates(m.sl,
                  old_state.subject_sl[incoming_line])
    then result.state = old_state with
      (.outgoing[outgoing_line] :=
        old_state.outgoing[outgoing_line] <: m)
      & result.decision = yes
    else result.state = old_state
      & result.decision = no
    fi);
end; { FILTER_Rules_of_Operation }

```

4.3. Proving the System Secure

The object of the scope `Filter_Rules_Model_Proof` is to establish that the rules of operation in the `Filter_Rules_of_Operation` and the system that they define are secure. It brings the pieces defined in various scopes together.

```

scope FILTER_Rules_Model_Proof =
begin
{ name declarations omitted }

```

The lemma `secure_FILTER` is the main theorem to establish. It demonstrates that the system defined by the set of `FILTER_operation_rules` is secure, as defined by `secure_system`. This is done by showing that if an `action_set` is created out of the rules and the initial state is valid as specified in the rules, then the resulting system is secure.

```

lemma secure_FILTER (z0: FILTER_state) =
  all wset: action_set,
  all rs: rule_set,
  FILTER_operation_rules(rs)
  & action_set_derived_from_rules(rs, wset, z0)
  & initial_FILTER_state(z0)
->
  secure_system(wset, z0);

```

The remaining theorems deal with the "auxiliary" security properties defined by the rule security properties. These must hold as well for the system's behavior to be truly restricted in the desired manner. The theorem `tranquility_preserving_FILTER` demonstrates that the FILTER Rules obey the model's tranquility property.

```

lemma tranquility_preserving_FILTER =
  all rs: rule_set,
  FILTER_operation_rules(rs)
->
  tranquility_preserving_rule_set(rs);

```

This theorem demonstrates that the FILTER Rules obey the model's buffer history preserving property. The rules specified obey constraints on how the buffer histories are changed

```

lemma buffer_history_preserving_FILTER =
  all rs: rule_set,
  FILTER_operation_rules(rs)
->
  buffer_history_preserving_rule_set(rs);

```

A rule set is the set of FILTER_operation_rules iff every rule in the rule set is one of the enumerated rules. Writing this theorem involves enumerating every rule of operation defined. FILTER has only one rule, so in this specific case there is no need to use a set of rules. We do so here in order to make this specification easy to generalize for an arbitrary number of rules.

```

function FILTER_operation_rules(rs: rule_set): boolean =
begin exit (assume result iff
  all r: rule,
  r in rs
  iff
  r = process_message_rule);
end;

end; { FILTER_Rules_Model_Proof }

```

5. SUMMARY

We have presented an approach for developing formal security models. The approach is in the style of Bell and LaPadula, to take advantage of user familiarity, but it is flexible enough to be adapted to a wide variety of state machine models. As in Bell and LaPadula, our approach consists of three steps: a **framework**, which defines the security policy; an abstract view of **system functionality**, which defines the rules of operation; and a **system security proof**, which proves that the rules of operation are consistent with respect to the security policy. Several examples of this approach have been written and proved completely within the Gypsy Verification Environment. As far as we know, these are some of the only examples of complete automated proofs faithful to the Bell and LaPadula style.

The insufficiency of a state invariant approach has been discussed by McLean[McLean87] and others. Our approach allows one to write further restrictions on state transitions to ensure a sound approach. The Gypsy specification in the last section contained lemmas called `tranquility_preserving_FILTER` and `buffer_history_preserving_FILTER`, which are

examples of such restrictions.

We hope that others will be interested in adapting this approach for their own use. The general Gypsy framework that we have provided will allow others to concentrate on the development of the specific security properties and rules of operation for their own system.

REFERENCES

- [Bell76] D.E. Bell, L.J. LaPadula, *Secure Computer Systems: Unified Exposition and Multics Interpretation*, MTR-2997 Rev. 1, MITRE Corp., Bedford, MA, March 1976.
- [Cohen86] R.M. Cohen, *Proving Gypsy Programs*, Institute for Computing Science, University of Texas at Austin, May 1986.
- [Criteria] *Department of Defense Trusted Computer System Evaluation Criteria*, 15 August 1983.
- [Good84] D.I. Good, *Revised Report on Gypsy 2.1*, Institute for Computing Science, University of Texas at Austin, March 1984.
- [McLean87] John McLean, *Reasoning About Security Models*, Proceedings of the 1987 IEEE Symposium on Security and Privacy.
- [Smith83] M.K. Smith, *Model and Design Proofs in Gypsy: An Example Using Bell and LaPadula*, Institute for Computing Science, University of Texas at Austin, February 1983.

TRUDATA: THE ROAD TO A TRUSTED DBMS

Ronald B. Knode

ORI/INTERCON Systems Corporation
9710 Patuxent Woods Drive
Columbia, Maryland 21046
(301) 381-9740

ABSTRACT

ORI/INTERCON Systems Corporation has encountered numerous mission needs for secure database management services offered in a practical, deployable, supportable system configuration. Our response has been to design a multi-level secure system that combines the contemporary architectural notions of security kernel, integrity lock, and trusted filtering and embark on an implementation program using existing products, technologies, and techniques. The INTERCON Trusted Database Management System (TRUDATA) project is targeted at an initial B1-level system with an ultimate B2 version as the eventual goal. This paper describes the "journey" that represents the TRUDATA project, including summaries of its development guidelines, system architecture, security policy, and implementation status.

WHY THE JOURNEY IS NECESSARY

Nearly every intelligence data processing and C3 system depends on the ready availability of accurate and timely data. Moreover, for "production cycle" intelligence environments, the quality of the product is typically fostered by the comprehensiveness of the data from which it was generated. The consequences are twofold:

- 1) Data must be accepted from a broad variety of sources.
- 2) Data must be retained and information captured over long periods of time.

The combination of these consequences results in collections of data of varying sensitivity. To date, the only way to provide a satisfactory data processing mechanism for such collections of data having multiple levels of security classifications is to dedicate a processing resource to the highest sensitivity of any data which might be in the collection, and then isolate that processing resource from all users who themselves do not have clearances to access all data potentially in the data reservoir. (The so-called "system high"

mode of operation.) As processing and reporting needs multiply, this technique forces duplication of processing power and data (to handle varying combinations of data sensitivity and user clearances) and prevents an effective flow of derived intelligence data to those users who have a subset of the clearances and accesses involved in the data pool. The net results are:

- 1) Expensive dedicated and duplicated intelligence data processing environments.
- 2) An inability to deliver or make available all the relevant intelligence and C3 product and data to all the users who legitimately need such data and are authorized to receive it.

WHY WE'RE MAKING THIS JOURNEY NOW

ORI/INTERCON has a long history of secure system development within the intelligence and C3 community. Pursuit of natural opportunities within those communities brought us face to face with several programs and procurements which had compelling requirements for MLS DBMS services. As we researched ways to solve these requirements, we found very little in the way of practical, existing foundations upon which to base a solution. After surveying the contemporary technological terrain, we decided that conditions were now right for our own expedition. Right from the start we determined that our destination was the delivery of trusted DBMS services in a truly useful implementation. In effect, we have three "passengers" on our journey, all of whom must reach the destination together in order for our trip to be successful. Our "passengers" are:

- 1) Doctrinal Security, i.e., TRUDATA must be secure in accordance with DOD policy and regulations.
- 2) Capability, i.e., TRUDATA must be capable of servicing real operational missions in a manner much like a conventional DBMS.

- 3) Achievability, i.e., TRUDATA must be doable using present-day products, technologies, and techniques.

Four conditions convinced us that the road to that destination could be traveled:

- 1) The Legacy of Previous Explorers

For over a decade now, researchers have been seriously exploring what it means to be a "secure DBMS". The landmark Air Force Summer Study at Woods Hole in 1982 ([3], [5]) summarized the "state of the problem" and projected three architectures that offered near term potential for supporting trusted DBMS services. Much additional work has been done since then to find productive paths to trusted database services (e.g., [2], [9], [17], [18], [19], [20]).

- 2) The New Traveling Vehicles

Computer system product advances that have occurred since the Summer Study make today the right time to move trusted DBMS technology beyond theoretical explorations to pragmatic implementations. These product advances represent new modes of transportation, i.e., new "vehicles", in which to travel the road toward trusted database management system development. Of particular importance are the "near" availability of certified secure versions of well known operating systems and the maturity of the database machine.

- 3) The Reward at Journey's End

Perhaps the most difficult traveling condition to gauge is the market for a trusted database management system. While we believe that such a product is not the next "Mustang" or "Taurus", we likewise believe that it is also not the next "Edsel" or "Corvair". However measured, our market surveys give us reason enough to follow this road to system development.

- 4) Good Traveling Companions

To succeed as a real solution to multi-level DBMS problems, a system must be deployable and supportable as well as offer secure DBMS services in convenient and efficient ways. Spurred on by this recognition, ORI/INTERCON Systems Corporation (INTERCON) has formed a Trusted Database Management System (TRUDATA) development alliance with Britton Lee, Inc. and AT&T to build and support a practical, deployable, functional Multi-Level Secure DBMS targeted ultimately at the B2-level

of certification. This combination of talent, products, and services gives us a ready-made support organization for the TRUDATA MLS DBMS.

TRAVEL PREPARATIONS

As for any journey, good preparations are crucial to reaching your destination on time and safely. Six main "traveling" guidelines have controlled our TRUDATA development project. In keeping with our journeying motif, we have paraphrased these guidelines as road-wise preparations:

- 1) Know what the end of the road should look like before you arrive; have good scouts to anticipate your arrival.

Interpretation

Have an implementation approach in mind from the very beginning. Make sure that implementation questions are resolved as security policy, system architecture, and assurance techniques evolve.

- 2) Use a good compass to stay aimed in the right direction.

Interpretation

Use a real operational mission archetype as a constant backdrop for development of architecture, policy, and actual implementation. Reconcile all design decisions to the fundamental question "Can the mission be fulfilled under this design?"

- 3) Watch out for "ambushes" from every direction.

Interpretation

Balance "theoretical" requirements against practical operational requirements and standard operational protections as a tradeoff mechanism in security policy development. Keep the policy from becoming so trivial that it is meaningless or so exotic that it is not implementable in some practical sense.

- 4) Don't stray too far from the beaten path.

Interpretation

Base the architecture on generally recognized and accepted techniques for achieving trusted DBMS services. Combine techniques to reduce or eliminate known vulnerabilities.

- 5) Start with a reliable mode of transportation.

Interpretation

Start with an already trusted product (operating system) as the basis for extending an existing TCB to support trusted DBMS services.

- 6) Stay in contact with your base camp and mark the trail well as you go.

Interpretation

Invoke the "social process" right from the start. Inform and involve the cognizant Government agencies and technical partners at every step. Document both the milestones (e.g., system architecture, security policy) and the process used to reach those milestones.

THE ROAD SO FAR

We've followed our own six "traveling tips" as we've moved down the road to TRUDATA implementation. For example:

- 1) Market surveys and our own secure systems development experience provided implementation characteristics (e.g., functionality, performance, size, cost) for TRUDATA from the very beginning.
- 2) We adopted the Naval Surveillance System (NSS) operational model described in [1] and broadened in [4] as our mission archetype.
- 3) We have invoked a process of constant self-examination to make sure that our security policy supports the mission archetype and abides by reasonable interpretations of [6] as amplified by the most recent successful research results.
- 4) We've constrained ourselves to generally accepted (though not yet officially sanctioned) trusted DBMS techniques and practices in design and development while still allowing ourselves the necessary freedom to innovate wherever creative compromises must be made.
- 5) From among a number of architecturally possible candidates, we have chosen AT&T's System V/MLS secure UNIX as the basis for the initial TRUDATA TFE and the Britton Lee IDM as the initial TRUDATA DBMS component. Crucial to these choices was an

acknowledged TCB foundation for TRUDATA TFE functions, and the intuitively appealing capability for physical as well as logical encapsulation of the DBMS.

- 6) Major TRUDATA project milestones ([7], [8], [22]) have been produced, reviewed, and presented to industry and Government from the very outset. These papers describe not only the "product" but also the continuing "process" used to take the next development steps.

Progress along the road to a TRUDATA implementation has occurred in short bursts of "breakthrough" speed interspersed with periods of "idling in place". The major milestones of architecture specification [7], security policy description [8], and implementation planning [22] have each been followed with a period of technical reflection and review, market reaffirmation, and project consolidation to march forward to the next milestone. As we proceed through our implementation plan, we anticipate a continuation of the "speed up-then-slow down" (and maybe even backup!) cycle as we attempt to navigate the specifics of an implementation without the benefit of an "official roadmap", i.e., Trusted DBMS Evaluation Criteria. Summaries of each of the major milestones leading up to actual implementation are provided in this survey of our journey to date. Full descriptions may be found in the references.

TRUDATA ARCHITECTURE SUMMARY

TRUDATA combines the classical "Integrity Lock" and "Kernelized" architectures from the Summer Study with an additional "Trusted Filtering" notion described in [9] to form the basis for the TRUDATA System Architecture [7] as shown in Figure 1. Two main architectural components provide the necessary functionality:

- 1) The TRUDATA architecture concentrates all "trust" for data and process security in a Trusted Front End (TFE) component.
- 2) The DBMS component (DBM) is completely encapsulated, making the only avenue of access to data management services through the TFE.

While we allow the DBM component to maintain data integrity and perform typical DBMS services (e.g., query resolution, data update), we do not trust it to do so in accordance with the TRUDATA security policy. The integrity locking technique, performed entirely in

the TFE, imparts that "trust" to the DBM by "checking up" on the integrity and accuracy of all data entering and leaving the TFE. A trusted filter, also in the TFE, either eliminates fundamentally non-secure DBM function requests before they are sent to the DBM or exchanges them for equivalent secure function requests ("commutative filtering"). Finally, additional trusted user interface software in the TFE provides for the secure administration and operation of TRUDATA. The TRUDATA system interface is SQL.

a priori, no system-driven classification attempts need be made. By making mviews "read-only", difficult update classification decisions can be avoided without sacrificing the real utility of data fusion in a database application.

- 3) A clearance vector for users (subjects) which can be used to enforce data classification control without constraining operational personnel to an unworkable disjointed scenario of partial data construction and review.

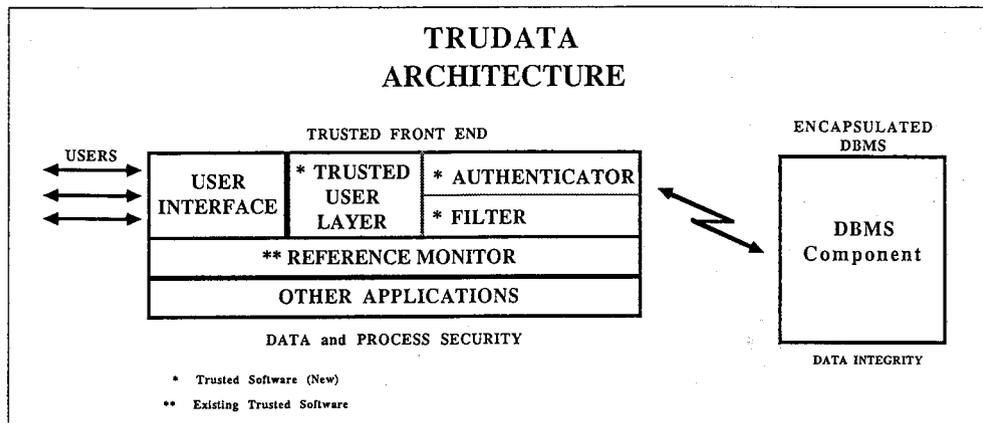


Figure 1

TRUDATA SECURITY POLICY SUMMARY

The TRUDATA Security Policy Model [8] is a derivative of the Naval Surveillance System (NSS) Security Model described by Graubart and Woodward in [1]. While we have adopted many of the same definitions and subpolicies espoused in the NSS model, the TRUDATA policy is distinguished by several new and different concepts, each of which helps to maintain a theoretically consistent model while at the same time supporting a practical concept of operations. Among the more important features of the model are:

- 1) Adjustable view level security. By making data access possible only through pre-defined views, TRUDATA permits the Data Base Administrator (DBA) to supply variations of security protection granularity extending from record level security (wherein the only defined view is that for the entire record) to field level security (in which single field views are defined).
- 2) The introduction of specially constrained view types called "pviews" and "mviews" as vehicles to control the inferencing and aggregation problem both within and across records. By forcing all views to be defined and assigned a default security level

Entities. The TRUDATA security model consists of five types of entities:

- 1) subjects
- 2) objects
- 3) security levels
- 4) operators
- 5) security policies

Subjects. The subjects in this model are the users. Each user is assigned a clearance vector $\langle d, u \rangle$ which establishes clearance level boundaries. The u component is the maximum clearance level of the user and the d component is the minimum clearance level of the user. Each user of the system is also assigned an access level $\langle a \rangle$. This level is the current security level of the user and must always satisfy

$$d \leq a \leq u$$

Operator Authorization List. In addition to clearance levels, there is also an operator authorization list (OAL) associated with each user. Only operators on this list can be invoked by

the user. Consequently, OAL's are used to help define the role within which each type of user must behave.

Objects. The objects in this model are:

- 1) data bases
- 2) relations
- 3) records (also called tuples)
- 4) views
- 5) fields (also called attributes)

Databases contain relations, which contain records, whose contents (i.e., field values) are available only through views.

Views. Views are named collections of fields within one or more relations. There are two kinds of views. Primitive views, called pviews, consist only of fields from a single relation, i.e., pviews are a projection from within a relation. Multiviews, called mviews, consist of a join of pviews within a database. To illustrate, consider two relations in a HOSPITAL database, one for PATIENTS and one for LABWORK. These relations are shown pictorially in Figure 2.

Likewise, an mview could be defined as:

```
3) create mview diagnosis
   as select ptag from
   patients,
   xray from labwork
   where
   patient_num = patient_num
```

All pviews for a relation must be defined at the time the relation itself is defined. Every relation has at least one pview, called the baseview. The baseview of a relation is the universal projection of all fields defined for the relation and is, therefore, equivalent to the relation itself in content definition. There is, however, one significant distinction. Namely, the baseview can behave as a container within a relation, providing a vehicle to control the view aggregation problem in much the same way as relation containers control the record (baseview) aggregation problem and database containers control the relation aggregation problem.

Protection Granularity. With our definition of views, we can now catalogue TRUDATA objects as either containers or atoms. TRUDATA containers are databases, relations, and baseviews/records. Views (including baseviews) are TRUDATA atoms because they are the smallest unit of information in the system to which

PATIENT Relation

Lastname	Firstname	Patient_Num	...
----------	-----------	-------------	-----

LABWORK Relation

Lab_Date	Patient_Num	Filmid	Technician	...
----------	-------------	--------	------------	-----

Figure 2

From the relations in Figure 2 pviews would be defined as:

- 1) create pview ptag


```
as select lastname,
   firstname,
   patient_num
   from patients
```
- 2) create pview xray


```
as select filmid,
   technician,
   patient_num,
   lab_date
   from labwork
```

explicit classifications are attached. Two interesting and helpful results occur naturally under this definition:

- 1) Baseviews behave as both containers and atoms, depending on their use/purpose within an application. Whenever whole records (with their field values) are legitimately accessed via the baseview, the baseview operates as an atom by providing an explicitly labeled collection of data. Likewise, whenever subcollections of data within a record are

legitimately accessed via other views, the baseview operates as a container by providing the superimposed mandatory access control service (the container clearance requirement) in the same way that database and relation containers perform.

- 2) Atomic level protection granularity is configurable by the DBA all the way from "record level" security (with only a baseview defined) to "field level" security (with single field pviews defined). Note that fields are in some sense "sub-atomic" units of information because they are only accessible through a view. Note further that field and view level security collapse into the same thing (an "atomic fusion" of sorts) whenever single field views are defined.

Security Levels.

Labels. TRUDATA security labels consist of both a hierarchical component and a set (possibly empty) of non-hierarchical categories. Subjects and data access objects (i.e., views, baseviews/records, relations, and databases) are labeled. For subjects, labels represent the clearances held by the subject. For data access objects, labels represent the classification of the object. Each coordinate of the subject clearance vector consists of a TRUDATA label.

All data access objects are labeled with an actual security level (ASL). A default ASL must be provided when data access objects are defined. Subsequently, the ASL is either explicitly provided when an instance of a data object is created (e.g., a record is created) or the defined default ASL becomes the ASL for the instance, i.e., the defined default ASL is inherited by the actual instance of the data access object.

Except for pviews, the default ASL of a data access object can be less than, greater than, or equal to that of its container. The ASL of a pview must always be less than or equal to that of its baseview container (since, in fact, such a restricted view always provides less data access than that of its baseview container). Such a restriction eliminates possible conflicts between data access to a baseview and to some pview within that baseview container.

Container Protection. The TRUDATA model enforces a container clearance requirement (CCR) for all data access. [Note: This is a uniform application of the Container Clearance Required notion from Landwehr's MMS model described in [17].] In addition to a default ASL,

each container is labeled with a Container Clearance Requirement (CCR) label and access to any data within a container is only allowed if the access level of the subject satisfies the non-discretionary access policies as applied to the CCR label. Thus, the CCR represents a "minimum" clearance level to be satisfied before access to any data in the container is allowed. For example, the CCR can be used on a baseview to insist that users have a Secret clearance before seeing any data via otherwise authorized pview(s), even though all pviews may only be labeled as Confidential. Such a capability supports control over "horizontal aggregation" (multiple views within a record) as well as "vertical aggregation" (a single view across many records).

The CCR label has no relationship with the default ASL of a container. The CCR may be less than, greater than, or equal to the default ASL. Furthermore, in recognition of the exceptionally strong tie between the definition of the relation and the automatically associated baseview container, the CCR of the relation is also defined to be the CCR of the baseview.

Access Control Lists. The security levels and labels described above form the basis for mandatory data protection within the TRUDATA security model. Discretionary security is represented via the concept of access control lists for each data access object. Access Control Lists (ACL's) can be associated with each data access object, and then can be used to determine what kind of access each subject can have to that data access object. Each entry on an ACL consists of the subject (or group) identifier and the access permissions authorized. ACL's control five types of access:

- 1) Access to Containers
- 2) Access to Relations
- 3) Access to Databases
- 4) Access to Non-Containers
- 5) Access to Pviews

ACLs list the exact access authorizations permitted to each individual/group specified on the list itself. Objects also have Exception Lists (EXL), which perform the opposite function. I.e., EXL's list all subjects for whom specific access authorizations are denied.

Operators. Users act upon data with operators. There are three classes of operators in the TRUDATA security model:

- 1) Those that access data.
- 2) Those that define data.

- 3) Those that manipulate the mandatory and discretionary access attributes of data.

Subjects can only use those operators that are listed in their OAL. The range of applicability of each class of operator is shown in Figure 3.

3) Discretionary Access Control

Discretionary access control is enforced using ACLs and EXLs. The ACL/EXL of the object, as well as the ACL/EXL of the object's container(s) if necessary, must authorize (and not exclude) the subject's requested operator access to the object.

The security policies define the relationships among users, operators, and objects. Consequently, the policies are described in [8] as they apply to each class of operator, i.e., data access, data definition, and attribute changing.

Operator Classes and Applicability

Operator Class	Operator Name	Operator Applicability
Data Access	Read Write Delete	All data access objects All data access objects except mviews Containers only
Data Definition	Define-db Define-rel Define-view	Databases only Relations only Relations only
Attribute Changing	Change-level/Read Change-access	All data access objects except mviews All data access objects

Figure 3

Policies. Subject to object access is controlled under a combination of three sub-policies. The requirements of all three sub-policies must be met before any specific access operation is allowed.

1) Operator Authorization.

Subjects access objects with operators. A subject can invoke an operator only if that operator is on the subject's OAL.

2) Mandatory Access Control

Mandatory (non-discretionary) access control involves the subject's security level, the ASL of the object, and the CCR of the object's container(s). The subject's security level must be sufficient to satisfy the mandatory access policies applied to the object for the given operator.

Added DBA Responsibilities. The TRUDATA security policy adds responsibilities to the role of the DBA and the system operator, primarily to inspect and maintain data integrity.

A Word About Integrity and Inference

Conflict Between Secrecy and Integrity. Much recent research [8,18,19,20] has led time and again to the frustrating realization that an inherent conflict exists between data secrecy and data integrity in database management systems. Data secrecy policies and data integrity policies are fundamentally orthogonal in motivation and practice. Inclusion of broad integrity subpolicies inevitably leads to a spiral of unacceptable covert and inference channels.

Current TRUDATA Approach. We, too, have confronted this situation in our effort to move generally accepted notions of trusted DBMS service from theory to reality with the application of careful

engineering judgment to a sound system architecture against a backdrop of real operational needs. We are concerned with data integrity issues as well as data security issues. The selection of a proven DBMS component with an extensive history of performance and an impressive capability for preserving data consistency reassures us that TRUDATA will be able to maintain database service. The available consistency tools and mechanisms for recovery bode well for TRUDATA as a reliable as well as trusted DBMS. In addition, the healthy set of access controls included in the TRUDATA security policies represents a strong contemporary approach to controlling inference.

Basis for Implementation Decisions. Yet, like [18], we continue to base our implementation decisions on the premise that "the security policy has precedence over, and a prior existence to, the integrity policy". Therefore, wherever the introduction of "supporting" policies such as integrity policies would reduce the ability of TRUDATA to enforce its security policy, we have ruled in favor of security policy enforcement as long as the operational problem archetype does not suffer. We expect that we will ultimately be able to reconcile additional integrity policies (e.g., constraints, rules) to our security policies. Through our own discoveries, as well as other ongoing research such as that reported in [20], we anticipate that some reasonable, implementable mechanisms will be identified. We have allowed for their eventual inclusion beyond our Phase I implementation program. Initially, however, database integrity will be a large responsibility of the SDBA. The SDBA must recognize when integrity considerations (constraints) impact data organization and classification, and then manifest that recognition with the right Container Clearance Requirement (CCR) and default Actual Security Level (ASL) selections (so that there is no newly introduced inference channel).

Sufficiency of Current Policy. We agree with [19] that it is unclear whether any integrity policy for DBMS's must be mandated at all. In view of the evolving consensus that a DBMS interpretation of [6] should not include requirements for controlling inference (ref [21]), we believe that our current policy direction satisfies the needs for an MLS DBMS implementation ultimately targeted at the B2 level, while at the same time providing for supporting policy growth as new practical possibilities emerge.

TRUDATA IMPLEMENTATION SUMMARY

Initial Configuration

Based on our TRUDATA operational requirements for a deployable, supportable system, an already trusted

operating system foundation, and a readily encapsulated DBMS capability, we have chosen to configure our initial TRUDATA system with an AT&T 3B2 Model 400 running System V/MLS as the TFE component and a Britton Lee IDM as the DBMS component. We are following our B2 level system target implementation plan, even though the current version of System V/MLS is in evaluation for B1 level certification. Our instant target is for an initial MLS TRUDATA at the B1 level, with an ultimate version (using even more secure versions of our baseline TFE operating system) targeted for B2. Britton Lee database machines provide the added reassurance of physical as well as logical encapsulation, plus high performance, a relational data model, and an existing support organization.

Implementation Issues

No Official Criteria. The absence of an "official" set of security criteria for trusted database management systems introduces an extra element of ambiguity into certain implementation choices. Much effort is currently being directed at discovering exactly what it means to be a "secure DBMS" certifiable to any specific level of trust. We, too, are participating in that effort as we prepare a TRUDATA which balances mission-based functional and performance imperatives with security policies that abide by generally accepted (if not formally sanctioned) DBMS interpretations.

Making and Tracking Implementation Choices. The National Computer Security Center (NCSC) is currently attempting to coalesce a set of criteria for trusted DBMSs. However, even if such a set of criteria were extant, the nature of a secure system implementation would still leave some of the thornier implementation choices unresolvable without experimental data and/or extensive collaborative deliberation. Certain semantic choices and covert channel bandwidth control choices are especially appropriate to this category.

In response to this situation, we have chosen to institute a "living" document, our TRUDATA Implementation Issues (TISSUES) List, around which we focus specific decision making efforts as we attempt to resolve (and document) every judgment issue discovered during implementation. We expect the TISSUES List to grow and shrink over time as implementation choices are made. Our first version of the TISSUES List had 14 different TISSUES to resolve.

Implementation Schedule

TRUDATA implementation is proceeding according to the TRUDATA Implementation Plan. The first phase of implementation is scheduled to occur in two stages.

Stage 1 is scheduled to end in the winter of 1987 with a prototype. Stage 2 finishes with our initial B1 target version in mid 1988. Subsequent phases are anticipated to move to a B2 targeted level and to install more user support tools, on-line expert Secure Data Base Administration (SDBA) guidance, and more refined supporting policies.

Implementation Procedure

TRUDATA is being implemented in a "closed security environment" according to National Computer Security Center guidance in [16]. After establishing the TRUDATA development facility and TRUDATA Configuration Management Plan and procedures, implementation is proceeding according to the following pattern:

- 1) "Absorb" the Britton Lee Portable Host Interface (PHI) source code and place under TRUDATA CM. "Absorption" includes a line-by-line inspection of all existing code to check for Trojan Horses and trapdoors.
- 2) Activation of the TRUDATA Assurance Program (TAP) consisting initially of rigorous configuration management, a continuous test program, and formal model interpretation. The TAP will be supplemented with covert channel analysis after Stage 1 is complete.
- 3) Confirmation of the standard Britton Lee UNIX PHI software in the System V/MLS version.
- 4) Insertion of the new "trusted authenticator" software at the system interface level of the PHI software. Careful examination of references [10] through [15] and analysis of PHI architecture as documented in [22] has isolated the points of protection to just a handful of routines (which must now be trusted).
- 5) Addition of Trusted User Services and Trusted Filter software.
- 6) Completion of data delivery services software.
- 7) Completion of Security Features User's Guide and Trusted Facility Manual.

Stage 1 is complete after step 4. The remaining steps complete Stage 2. Superimposed over the entire pattern is:

- 1) A program of periodic reporting and review with development partners and involved Government agencies.

- 2) The maintenance of a "living" document (the TISSUES List) which tracks implementation issues and their resolution throughout the implementation process.
- 3) Concurrent development of an application scenario as a way of confirming our implementation decisions and demonstrating trusted DBMS services.

REFERENCES

- [1] A Preliminary Naval Surveillance DBMS Security Model, Graubart, R.D., Woodward, J.P.L., Mitre, 1982.
- [2] Integrity Lock Designs Document, MTR9505, Duffy, K.J. Graubart, R.D., August 1985
- [3] Multilevel Data Management Security, Committee on Multilevel Data Management Security, Air Force Studies Board, Commission on Engineering and Technical Systems, National Research Council, National Academy Press, 1983.
- [4] Security Requirements of Navy Embedded Computers, NRL Memorandum Report 5425, Carroll, J.M., Froscher, J.N., 28 Sept 1984.
- [5] Survey of Technology Applicable to the Design of Multilevel Secure Database Management Systems, System Development Corporation (for RADC), November 1983.
- [6] Trusted Computer System Evaluation Criteria, NCSC-STD-001-83, 15 August 1983.
- [7] TRUDATA: ORI/INTERCON TRUSTED DATABASE MANAGEMENT SYSTEM, System Development Plan, System Architecture, Knode, R. B., ORI/INTERCON, 3 October 1986.
- [8] TRUDATA SECURITY POLICY MODEL, A Descriptive Top Level Specification, Knode, R. B., ORI/INTERCON, 26 November 1986.
- [9] Commutative Filters for Reducing Inference Threats in Multilevel Database Systems, Denning, D. E., SRI International, Proceedings of the Symposium on Security and Privacy, 1985.
- [10] IDM Software Reference Manual, Britton Lee, Inc., Part Number 202-0500-019, June 1986.
- [11] Portable Host Interface Software

- Specification, Britton Lee, Inc.,
Part Number 205-1190-007,
January 1986.
- [12] IDMLIB User's Guide, Britton Lee,
Inc., Part Number 205-1681-000,
July 1986.
- [13] IDM 500 Operation Manual,
Britton Lee, Inc., Part
Number 201-1078-006, June 1986.
- [14] IDM 500 Maintenance Manual,
Britton Lee, Inc., Part
Number 201-1210-002, June 1986.
- [15] Predefined Stored Commands,
Britton Lee, Inc., Part
Number 205-1607-000,
February 1986.
- [16] Computer Security Requirements.
Guidance for Applying the
Department of Defense Trusted
Computer System Evaluation
Criteria in Specific
Environments. CSC-STD-003-85.
National Computer Security
Center, 25 June 1985.
- [17] A Security Model for Military
Message Systems, ACM Transactions
on Computer Systems, Vol. 2,
No. 3, pp.198-222.
Landwehr, C. E.,
Heitmeyer, C. L., and McLean, J.,
August 1984.
- [18] A New Look at Integrity Policy
for Database Management Systems.
Proceedings of the National
Computer Security Center
Invitational Workshop on Database
Security (17-20 June 1986).
Bonyun, D., I.P. Sharp Associates
Limited, June 1986.
- [19] Integrity in Trusted Database
Systems, Proceedings of the
National Computer Security Center
Invitational Workshop on Database
Security (17-20 June 1986),
Schell, Roger R. (Gemini
Computers), Denning, Dorothy E.
(SRI International). June 1986.
- [20] Secure Distributed Data Views,
Security Policy and
Interpretation for a Class A1
Multilevel Secure Relational
Database System, Denning, D. E.,
et alia (SRI International),
Schell, Roger R., et alia
(Gemini Computers),
November 1986.
- [21] Integrity and Inference Group
Report, Proceedings of the
National Computer Security Center
Invitational Workshop on Database
Security (17-20 June 1986).
- [22] TRUDATA IMPLEMENTATION PLAN,
Knode, R. B., ORI/INTERCON,
27 February 1987.

THE SYBASE SECURE DATASERVER: A SOLUTION TO THE MULTILEVEL SECURE DBMS PROBLEM

Patricia A. Rougeau
Edward D. Sturms

TRW Federal Systems Group
2751 Prosperity Avenue
P.O. Box 10440
Fairfax, VA 22031

INTRODUCTION

Today's database management system (DBMS) technology is severely limited in its ability to protect sensitive information and meet the increasingly demanding performance requirements of many government, military, and private sector data processing systems. Current high performance DBMSs do not offer data security, and previous secure DBMS prototypes suffered in their performance, flexibility, and maintainability. The Sybase Secure DataServer (SYSDS) effort intends to solve both the security and performance problems associated with modern, relational DBMSs.

This paper presents the SYSDS approach to solving the secure DBMS problem. The SYSDS is a multilevel secure relational DBMS, based on the Sybase relational DBMS known as the DataServer*. The SYSDS is currently under development. The original SYSDS approach took advantage of the fact that the DataServer was in an early development stage. The current DataServer represents a state-of-the-art relational data management system which when modified, yields a cost-effective, reliable multilevel secure DBMS that does not sacrifice essential performance characteristics.

THE TRUSTED DBMS PROBLEM

In 1982, the Air Force Studies Board stated that computer security technology had advanced to the point where certifiable multilevel secure DBMSs could be built in the near term [AFSB83]. However, this technology has not materialized in the commercial marketplace.

The SYSDS addresses several problems confronted by designers of multilevel secure database management systems. These include:

- Storage of multilevel data
- Data and system integrity
- Performance
- Design Criteria
- Technological Obsolescence.

Storage of Multilevel Data

Most commercial operating systems that attempt to provide mandatory and discretionary security controls do so at the file level. This is insufficient for many applications, particularly in a

military command and control environment where the granularity of access protection must be very fine. The SYSDS design provides mandatory protection at the row level, with up to 16 hierarchical classifications and 64 non-hierarchical categories.

Data and System Integrity

The *DoD Trusted Computer System Evaluation Criteria* (the Criteria), the governing document behind computer security, does not address the problem of data integrity, a problem particularly applicable to database management systems [DODT83]. The SYSDS approach addresses this problem in three ways. First, protection against inadvertent errors, such as hardware problems, is provided by the use of an integrity field covering every data page. This integrity field contains an error detection code called a cyclic redundancy check (CRC). The CRC is used for integrity purposes, not for security purposes, since the SYSDS is a reference monitor approach. Second, the SYSDS interfaces with network encryption devices on output for secure end-to-end transmission of data over untrusted networks. These two methods provide data integrity both within the SYSDS and between cooperating hosts. Third, the SYSDS introduces the concept of Trusted Computing Base (TCB) integrity by separating trusted code into two hardware supported execution domains to help limit the amount of trust afforded to each domain. This unique approach provides system integrity.

The SYSDS offers other DBMS integrity features not included in the TCB. For example, the SYSDS enforces range checks and triggers, but these mechanisms are not enforced via trusted code. They were intentionally left out of trusted code to help reduce the size of the TCB. Placing them in trusted code would have meant including a substantial portion of the SQL Compiler in the TCB, making the TCB significantly larger. In essence, this corrupts the purpose of a reference monitor approach since the reference monitor would no longer be small enough to verify.

Performance

One of the largest problems in the construction of secure systems is that, whether the system is an operating system or a secure DBMS, security controls often degrade system performance to the point that the system no longer meets operational requirements. This renders the system secure but impractical for the mission or application. The result is that security controls are turned off or compromises are made and organizations are forced to purchase unsecure systems that better meet the performance requirements of the application. System users will tolerate some performance degradation due to security, but it must be minimal.

*DataServer is a trademark of Sybase, Inc.

The SYSDS, as noted, is based on an entirely re-architected DataServer. This approach is unique in that the DataServer was designed with performance in mind, yielding an advantage in modifying the system to meet Class B2 requirements as specified in the Criteria. The SYSDS was designed without introducing excessive risk and performance penalties. At the time of the original SYSDS design, the DataServer itself was in an early development stage making it amendable to the type of changes needed for security - changes that have proven very difficult to implement in existing commercial DBMS products.

The SYSDS design takes full advantage of high performance features found in the Sybase DataServer. A primary goal of the SYSDS effort was to modify the DataServer design, adding security mechanisms while preserving the features which provide high performance. To help meet this goal, the SYSDS will run on a bare machine, making it a secure, high performance database machine.

Design Criteria

The *DoD Trusted Computer Systems Evaluation Criteria* is the guiding document governing the design and development of secure systems. Unfortunately, this document was originally intended to serve the needs of operating system designers, and in some cases, cannot readily be extended to govern the construction of database management systems. To correct this situation, the National Computer Security Center is developing a set of Trusted DBMS Guidelines. In the absence of Trusted DBMS Guidelines, the Criteria must be applied.

According to the Criteria, applications which use labeled data must address the B Division requirements. Specifically, the SYSDS intends to meet the Class B2 level requirements, with the addition of special integrity mechanisms.

Current thinking in the security community indicates that a database management system must be evaluated together with the operating system on which it resides. The SYSDS approach alleviates this problem — there are very few B2-level secure operating systems — by residing on bare hardware without the support of a commercial operating system. All operating system functions are part of the DBMS kernel.

Technological Obsolescence

Because of the delays inherent in secure system development, many secure systems fall behind the state-of-the-art by the time they reach the prototype phase. In other words, they are obsolete shortly after proof of concept. The SYSDS approach has no definitive answer to this problem, but a growth-path has been designed. The SYSDS will be based on the commercial Sybase DataServer, allowing enhancements made to the non-secure system to be applied to the secure version on a case by case basis. Of course, the SYSDS will have to be re-certified after every major update, but there are plans for revisions in order to keep the SYSDS current with the state-of-the-art in database management systems.

THE SYSDS SOLUTION

Designers of trusted DBMSs have difficulty in establishing what software needs to be trusted and what can remain untrusted. In addition to the mandatory and discretionary policies addressed

in the Criteria, secure database designers must address special integrity issues, including domain integrity (e.g. range of values) and relational integrity (e.g. referential integrity). Security considerations are further complicated by the wide range of architectures available to the DBMS designer, from database systems executing on top of a target operating system to back-end database machines using no commercial operating system at all [HENN86].

In addressing the security and integrity concerns that necessitate the TCB, there is a tendency to allow the TCB perimeter to grow until it encompasses the majority of the DBMS code. If this happens, the DBMS TCB no longer satisfies the reference monitor concept stated in the Criteria since it will not be small enough to verify. Schell and Denning addressed this problem by defining two TCB perimeters, one for mandatory security and integrity, and the other for discretionary security, recovery, etc. [SCHL86]. This enabled them to keep the mandatory security kernel small, but, since their primary security object was the data view, all of the semantic-related tools in the DBMS had to reside in the second perimeter. This, coupled with recovery code and other trusted mechanisms, could potentially make the second perimeter large.

In defining the perimeter of the SYSDS TCB, the following considerations were made. Since the SYSDS runs on a bare machine, the interrelations and dependencies between the DBMS and the target operating system did not have to be considered. Since the design of the commercial DataServer includes many integrity and discretionary control features, a decision had to be made as to which features to keep in the TCB and which features to remove so that the TCB would not be too large. Table 1 summarizes DBMS features inside and outside the TCB.

Design Feature	SYSDS TCB Implemented	Outside TCB
Login	•	
Auditing	•	
Trusted Recovery	•	
Mandatory Access At Record Level	•	
Discretionary Access At Table Level for the Operations Select, Insert, Delete, Upgrade	•	
Integrity CRC Checks	•	
Trusted Operations	•	
Range Checks		•
Triggers		•
View Protection		•
Deadlock Detection	•	
Livelock Detection	•	
Concurrency Control	•	

Table 1. Perimeter of the TCB

One option considered was to add mandatory security mechanisms to the original DataServer design and to have the entire DBMS become the TCB. Using version 2.0 of the non-secure Sybase DataServer as the basis of estimate, with 39,000 lines of code, this option was within the feasible verification range. SCOMP, certified as an A1 system, has approximately 35,000 lines of code, about 10,000 lines of Pascal code in the security kernel and another 25,000 lines of C code in the trusted software. Although it would have been easier to designate the entire DBMS trusted, this approach was rejected because it

meant that the TCB would contain considerable code not relevant to security and would be larger than necessary.

A second option, separating mandatory and discretionary security into two perimeters, was also rejected. Although the mandatory and discretionary mechanisms could be separated, retaining views as objects would have made the two domains, including both mandatory and discretionary controls, almost the same size as the first option — nearly the whole DBMS.

The approach chosen for the SYSDS was to define only one TCB perimeter to include mandatory security, discretionary security, integrity, recovery, auditing, and trusted operations. Most of the complex semantic-related code in the SQL compiler, was placed outside of the TCB. With this approach, the integrity features, such as triggers, are not included in the TCB. Instead, these features are available outside the TCB and can be used to augment or enhance the SYSDS security policy. The secure operation of the SYSDS does not depend on the correct use of these mechanisms. In this way, a single TCB embodies the essential features of the security approach, while remaining as small as technically possible.

THE SYSDS SECURITY MODEL

Subjects

In the SYSDS, subjects are active entities. A subject is defined as a process running on behalf of a user. A key aspect of the SYSDS design is that the database will be a stand-alone, dedicated back-end processor. Trusted software in the DBMS will create a user process in the machine for the duration of a user session. The user process is assigned the security level of the user at the time the process is created. Users are allowed to designate the security level of a session as long as the level does not exceed the maximum clearance of the user. The maximum clearance of the user is stored in the DBMS.

Objects

One of the major difficulties associated with applying the Criteria to database systems has been effectively defining the varying granularity of system objects. In the SYSDS model, an object is defined as one of two types of objects:

- Primary Object (PO)
- Secondary Object (SO).

A PO is defined as a data row (i.e., record or tuple) in a table. All POs are governed by the mandatory access policy and the CRC integrity control mechanism but not discretionary access policy. SOs are defined as databases and tables. All SOs are subject to discretionary access policy only; no mandatory access or integrity policy is directly applied to SOs.

The definition of PO and SO holds for all SYSDS objects, including system objects in the data dictionary. Since every row in the data dictionary is considered a PO, the SYSDS model adds the benefit of implementing a minimum security level on accesses to databases, tables, or even columns, regardless of the information contained in them. For example, a database may be designed to contain rows varying in classification from

UNCLASSIFIED to TOP SECRET. However, if the row in the data dictionary, which refers to the name of a database, is classified CONFIDENTIAL, then all users who reference that database must have a login-level of at least CONFIDENTIAL in order to gain access to that database. The same holds true for access to other system objects such as tables and columns. This SYSDS design feature can be used at the discretion of the database designer. If this feature is not necessary for a given application, the database designer can create all system objects at a system-low or UNCLASSIFIED level, meaning that each database user will at least gain access to the system object names subject to discretionary access checks, and mandatory access will then be checked at the row level of the base tables.

Defining two levels of system objects addresses both implementation efficiency and Criteria requirements. First, to ensure accurate security of system data, mandatory access protection must be applied to every data row accessed in the DBMS. This prevents a disclosure of data. However, it would be time consuming to also check discretionary access rules on a per-row basis. In most DBMSs, such checks are done at a higher level, usually the database and table level. The SYSDS maintains this traditional approach since it provides efficient and accurate discretionary protection of the data.

Second, the Criteria requirements state that all accesses to named objects in the system must be audited. Even with the number of rows in a small database and the potential accesses generated by a few users, this requirement could easily produce a voluminous and useless audit trail. In an effort to control this problem and still meet the Criteria, it is expected that only SO accesses will normally be audited in the SYSDS although the capability will exist to audit all successful accesses on a per table or per user basis. The SYSDS also allows actions to be audited on a per command basis. For example, it is possible to audit only UPDATE and DELETE operations on a specific table. Thus, the SYSDS can audit all accesses to SOs (i.e., down to the table level) and check mandatory access and integrity of all requested POs. Although the capability will exist to audit every access to each record, it is expected that only anomalies will be audited at the PO level. This approach controls access to multi-level data while meeting certain Criteria and application requirements.

Database Operations

As with any DBMS, the SYSDS has a set of common operations known as primary operations. Primary operations are performed directly against POs, although in the execution of each primary operation there is discretionary access validation for each operation for all SOs referenced in the operation. Only four of these operations are discussed here. The following paragraphs present an overview of these primary operations.

Select. The Select operation retrieves rows, or combinations of rows, from one or more tables in the database. Prior to selecting rows, the TCB validates discretionary access on all SOs (i.e., the database and the table) referenced in the selection criteria. Again, discretionary access is based on a per-command basis so it is possible to not have SELECT access to a particular table. The TCB also validates the security label of each row satisfying the selection criteria and retrieves only those rows dominated by the login-level of the subject.

Update. The update operation modifies one or more columns within an existing row. Prior to performing the update, the TCB validates discretionary access on the SOs referenced in the up-

date criteria. The TCB performs mandatory access validation on the row to be updated. For an update, the subject's login clearance must dominate the security label of the row to be modified. After the update, the row inherits the login level of the subject which performed the modification, and the TCB recalculates the CRC of the data page on which the row resides.

Insert. The insert operation places new rows into one or more tables. Prior to performing the insert, the TCB validates discretionary access to the SOs under consideration. Each new row inserted into the table inherits the login-level of the subject.

Delete. The delete operation removes existing rows from the table. The TCB validates discretionary access to the SOs referenced in the operations. The TCB also performs mandatory access validation on the rows to be deleted. Prior to the delete, the TCB will ensure that the security label of each row to be deleted is dominated by the login-level of the subject. Subjects are not allowed to delete rows to which they do not have access.

Integrity

As already mentioned, integrity is of primary concern to database management system designers. However, the Criteria does not present specific integrity guidelines. The SYSDS model addresses the problems of data corruption, i.e., the problem of data modification rather than data access. The SYSDS policy encompasses accidental modification, unauthorized modification, as well as integrity checking for the correctness of database data.

Biba has proposed several solutions to the integrity problem including his strict integrity policy, the low-water mark policy, and the ring policy [BIBA77]. Unfortunately, many other researchers have found these theoretical policies overly restrictive. For example, the strict integrity policy restrictions are unwieldy in application. If a program reads data of low integrity, it cannot write data of high integrity. This led Schell to propose a special case of the strict integrity policy in which read access and execute access are distinguished [SCHL86]. Boebert and Kain found that hierarchical integrity policies, which bind integrity levels to subjects as well as objects, are difficult to apply in application in that they have excessive reliance on "trusted" subjects [BOEB85]. Finally, Landwehr omitted integrity levels from the Military Message System security model because there is no mechanism in the government for accountability with regard to the protection of data against modification [LAND82, LAND84].

The SYSDS does not implement a hierarchy of integrity levels but rather addresses the concept of TCB integrity and correctness of data pages and security-relevant objects. The TCB is divided into two hardware domains, forcing an overlay of least-privilege on the code. The I/O Domain deals directly with all hardware elements in the system and is the only domain capable of altering the data base. The Policy Domain is the data base management engine.

Other DBMSs do not emphasize the correctness of data and the criticality of well-formed tables. The SYSDS uses the CRC to detect unintentional (or intentional) errors in data correctness. The CRC is calculated on a page basis. In addition to this internal CRC check, the system will return to the host a CRC calculated over the data row to assure the correctness of the row while in transmission.

Finally, the SYSDS uses trusted code to build security-relevant

tables such as the login account table, the user clearance table, and the discretionary access authorization table. No commercial system today can guarantee that data will not be inserted into or removed from an incorrect row. By using trusted code, the SYSDS makes it possible to make assertions as to the correctness of security-relevant tables, assertions which cannot be made if the tables are constructed by an untrusted SQL compiler.

THE SYSDS ARCHITECTURE

Hardware Architecture

The Digital Equipment Corporation (DEC*) VAX product line is the target hardware environment for SYSDS. It provides a compatible family of price/performance machines from a major manufacturer. Any machine with at least three hardware domains could have been chosen. In fact, if the internal division of the TCB into two parts had not been a goal, any machine with two domains could have been used.

All VAXs, from the MicroVAX through the 8850, have a memory architecture with four access modes or domains. The access modes are organized in a strict hierarchy which DEC calls User, Supervisor, Executive, and Kernel, going from least to most privileged (Supervisor mode is not used in the SYSDS architecture). To go from a lower privilege to higher privilege requires a system call. In this way, the TCB can control the call and all data accesses in the call. To go from a higher to a lower privilege domain can be done by a return from a system call.

Each page of memory in the VAX can be marked with the least privileged domain that can read it and the least privileged domain that can write it. For example, a page can be marked as read by User mode and write by Kernel mode, meaning that all four modes can read the page but only the Kernel mode can write the page. This mechanism is used extensively in the SYSDS. It is not possible, for example, to allow read access by Executive mode but not by Kernel mode since this breaks the strict hierarchy. The modes are used to provide separation of trusted and untrusted processes, as well as provide separation of functions within the TCB.

Software Architecture

The SYSDS software architecture is divided into three code bodies, each of which runs in its own hardware access mode of the VAX. The software is divided into one untrusted domain and two trusted domains comprising the TCB. The SYSDS software architecture maps directly into the four VAX access modes. Figure 1, SYSDS Software Architecture, illustrates the different domains.

The I/O Domain. The I/O Domain, executing in the most privileged Kernel access mode, is reserved for software that manages the hardware and directly manipulates the data on the disk, in cache, or on the network. Software in the I/O Domain is responsible for:

- Process Control
- Hardware Control
- Page Integrity.

*DEC, VAX, MicroVAX, VMS, and ULTRIX are trademarks of Digital Equipment Corporation.

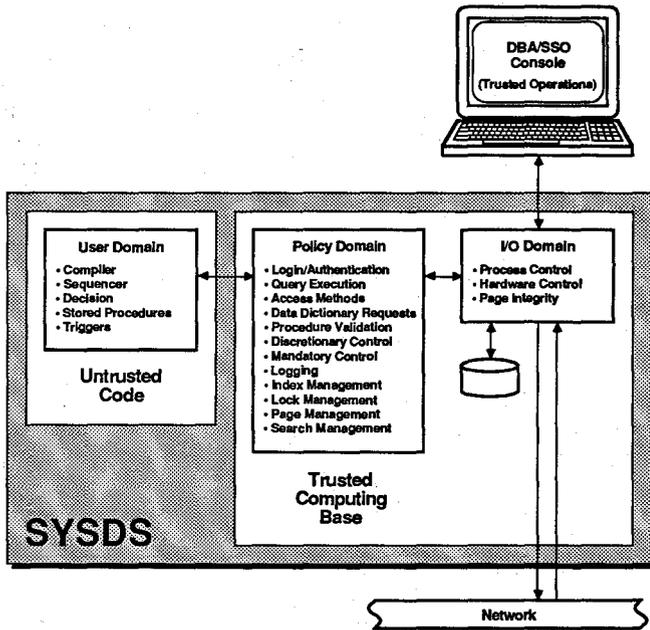


Figure 1. SYSDS Software Architecture

The I/O Domain replaces the traditional operating system. Since its only function is to provide a run-time environment for the database, its size is very small. Excluding device drivers, it is its size will be approximately 2,600 C statements. All estimates are based on Sybase DataServer 2.0 line counts of comparable code. The device drivers will be adapted from ULTRIX.

The I/O Domain is the only domain which has write access to the database cache. When a page is needed from the database (i.e., disk), it is read by the I/O Domain into a cache buffer in main memory. Each page has an ID and CRC on it used to confirm that the disk controller read the correct page and verify the correctness or integrity of the page itself.

The Policy Domain. The Policy Domain contains the entire security policy for the SYSDS and is the primary execution engine for the database. The Policy Domain also includes a library of subroutine services used mainly by code in the same domain. This library supports the management of indices, locks, pages, and searches. The Policy Domain runs in the Executive mode of the VAX and is the next highest privileged access mode after the Kernel. Code in the Policy Domain implements the following functional units:

- Authentication
- Query Execution
- Access Methods
- Data Dictionary Requests
- Procedure Validation
- Discretionary Access Control
- Mandatory Access Control
- Logging

- Index Management
- Lock Management
- Page Management
- Search Management.

User Domain. The User Domain translates and compiles SQL (the query language) statements into procedures which can be executed by the Policy Domain. All User Domain code runs in User mode on the VAX and is considered untrusted. User Domain code calls the Policy Domain via a system call and cannot call the I/O Domain directly. This code is nearly identical to the existing Sybase DataServer code that performs the same functions. The functional units in the User Domain include:

- The Compiler
- The Sequencer
- Decision
- Stored Procedures
- Triggers.

THE SYSDS IN OPERATION

Figure 2 illustrates a scenario tying all of this information together. After the Policy Domain has received a User ID, Password, and Login-level Clearance, and the user is logged in, an untrusted process is created by the Policy and I/O Domains on behalf of the user. From that point on, processing requests are received in the TCB and passed to the untrusted user process in the User Domain for parsing and compilation. For example, commands in the form of SQL statements are received from the host via a network. The I/O Domain is responsible for decoding the statements and passing them to the Policy Domain for dissemination. The Policy Domain in turn distributes the command to the correct user process executing in the untrusted User Domain.

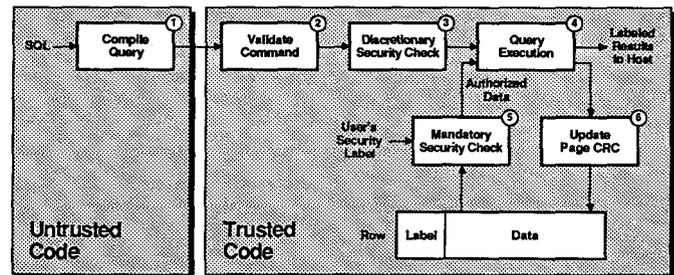


Figure 2. SYSDS Operations

After the untrusted user process receives the command, it first compiles the SQL statement(s) into a binary internal format called a Procedure, to be passed to the TCB for execution. The compilation step requires a great deal of code, and, in the SYSDS, it was determined that all of this code could remain untrusted, thus reducing the size of the TCB. The TCB handles all aspects of the execution of the Procedure after it has been compiled, including the retransmission of the results to the host.

Computer Security at Sun Microsystems, Inc.

Katherine Addison, Larry Baron, Mark Copple,
Don Cragun, Keith Hospers, Patricia Jordan,
Mikel Lechner, Michael Manley, Casey Schaufier

Sun Microsystems, Inc.
Mountain View, California

ABSTRACT

Sun Microsystems is currently developing enhancements to its Sun Workstation and SunOS products to create a Trusted Computing Base to be evaluated at the B1 level. In this paper, that product is referred to as "Secure SunOS". This paper describes the project's history, status, and goals, as well as discussing the more interesting aspects of the Secure SunOS product. This paper also describes some of Sun's future directions in the secure systems marketplace.

INTRODUCTION

In late 1985, Sun Microsystems established the Sun Federal Systems Division to do business in government marketplaces. It soon became apparent that computer security would play an important role in these markets, and that Sun would have to develop a Trusted Computing Base (TCB), based on its Sun Workstation and SunOS products, to be evaluated according to the requirements of the Trusted Computer System Evaluation Criteria [DoD85]. This work has been going on in earnest for a about one year, and the purpose of this paper is to describe what Sun has been doing and where Sun is going in the computer security marketplace.

The first half of this paper begins by describing the history of the Secure SunOS project, the near-term product plans for secure computing, development directions and goals, and Sun's work in UNIX system security standards. The second half of the paper describes the more interesting features of this initial version of Secure SunOS, which is targeted for the B1 TCSEC level. The description of Secure SunOS does not attempt to explain the underlying SunOS system (Sun's enhanced version of the UNIX system), the basic concepts of mandatory security, or the requirements of the TCSEC, because these topics have been covered well by other papers in the past.

CURRENT STATUS

Sun got started in the computer security business only about a year ago. The initial impetus was provided by government procurements with requirements for the C2 and B1 security levels defined in the TCSEC. It also became clear that it would be increasingly important to have an evaluated secure product, and moreover, security features that were flexible enough to apply in commercial environments as well as for government customers.

UNIX is a registered trademark of AT&T.

NFS and SunOS are registered trademarks of Sun Microsystems, Inc.

Xenix is a registered trademark of Microsoft.

Because B1 is the highest level readily achievable by commercial systems, it is most common in current government procurements. Although some procurements specify C2, a B1 system will satisfy any C2 (or C1) requirement, as well as all B1 requirements. Since the NCSC evaluation process is expensive and time-consuming, Sun decided to forego a C2 evaluation and have Secure SunOS initially evaluated for B1. The main consequence of this decision is that an evaluated version of SunOS will be delivered somewhat later than might have been possible had only the C2 features been added, but this additional delay is relatively small. Even though Secure SunOS will not be delivered this year, all major C2 features will be present (though unevaluated) in the next release of standard SunOS.

In keeping with Sun's commitment to develop and support standards for UNIX systems, Sun is working with several standards groups, and some other vendors, to settle on common definitions for security labels, password protection, auditing, access control lists, and so forth.

Adding basic C2/B1 security features to a UNIX system such as SunOS is straightforward. The challenge lies in making those features both powerful and sufficiently easy to use that they can be applied in many environments, not just that of federal government classified information processing. The system must also be designed in a way that does not conflict with user's expectations for a standards-conforming UNIX system. Sun is committed to producing technically advanced secure systems, with features beyond the relatively simple requirements of the Criteria. This is particularly important for the commercial and educational marketplaces, where mandatory access control mechanisms may not fit an organization's needs, but where increased security and administrative control are very important.

CURRENT PRODUCT PLANS

The primary focus of Sun's effort is the Secure SunOS product. This is a system intended to meet the B1 requirements of the TCSEC, and is currently nearing the end of its initial development cycle. The first result of this work is the package of "C2 Features" to be delivered in SunOS Release 4.0.

The C2 Features Package

The "C2 Features" package is primarily intended to give customers an early chance to experiment with the Auditing mechanism that will be fully implemented in Secure SunOS. The package also includes protection for user passwords, additional documentation, and some initial support for the labeling features in Secure SunOS. With the "C2 Features" package fully installed, SunOS Release 4.0 will satisfy all the major C2 requirements of the TCSEC. It was not submitted for evaluation primarily to save the effort of a full-scale security evaluation for the SunB1 product, and also because it is incomplete in minor areas.

This proved to be a wise decision, because it allowed us to put many of the underpinnings of Secure SunOS into place much earlier than would otherwise have been possible. It also allows us a good chance to tune the audit mechanism and make it easier to use; since the "feel" of an audit facility is very difficult to evaluate without experience using it, this is very important.

The Secure SunOS Product

Secure SunOS is an independent product, derived from the current version of SunOS, but developed and tested separately. Eventually, the security features in each Secure SunOS release are expected to become part of standard SunOS. Initially, however, it is a separate product to ensure that the NCSC evaluation process runs as smoothly as possible: since the main goal for Secure SunOS is B1 security, it can be changed during the evaluation process much more easily than SunOS, which must respond to a wide variety of requirements. Having a separate Secure SunOS product also allows us to coordinate product release with the formal evaluation, rather than being tied to the regular release schedule. Nonetheless, it is really just a version of SunOS, and the two products are kept closely tied.

The goals of Secure SunOS are:

- Conformance with NCSC B1 criteria
- Keeping the UNIX "feel" in a secure system
- Useful in commercial as well as government applications
- Compatibility with the standard SunOS specification
- Minimal change to standard SunOS interfaces
- Operation in standard Sun network environment
- Extensibility to support additional security features

The B1 level was chosen to meet the dual goals of satisfying a large number of procurements and providing a product in a reasonable timeframe. Although Sun is considering higher TCSEC levels, the initial emphasis is on security features rather than internal assurances, on the assumption that the customers need to gain experience with those features before they will know what they really want in a more secure system. Secure SunOS therefore includes some of the security features required at B2 and B3: device labels, realtime audit alarms, etc., even though it is only being evaluated for B1. The security policy and descriptive top-level specification are also written to satisfy the B2 requirements.

FUTURE DIRECTIONS

In general terms, Sun will enhance Secure SunOS in response to customer requirements, but there are some specific plans for work in the areas described below.

Standards

Sun is working with other vendors and the NCSC towards a Secure UNIX system standard. The standards work is being done under the auspices of the P1003.1 Portable Operating System for Computer Environments working group of the IEEE Computer Society Technical Committee on Operating Systems and the /usr/group Technical Committee Subcommittee on Security. The latter group is responsible for preparing all security-related inputs to the IEEE POSIX committee.

Network Security

Because the Sun system is a distributed collection of workstations and servers connected by a network, Sun has a more pressing interest in network security than many other vendors. The basic SunOS architecture allows a collection of workstations to appear as a single, distributed, TCB. Therefore, for evaluation purposes, an entire Secure SunOS configuration is considered as a single "system", all of whose hardware must be physically secure. Even in environments where hardware is not wholly secure, the "secure booting" mechanism described below will provide sufficient protection in many environments.

In addition, although the NCSC evaluation will not cover such configurations, Sun plans for Secure SunOS to operate compatibly when connected to networks containing non-Secure SunOS machines (either Sun machines or others). Non-Secure SunOS machines will be treated as single-level systems. Finally, all Secure SunOS systems will be able to use the "secure networking" mechanisms in SunOS, which use public-key encryption to ensure secure authentication even if the network contains untrusted hardware. Again, in the NCSC-evaluated B1 environment, this will be unnecessary because the hardware itself must be secured, these features will be useful in other environments.

Compartmented Mode Workstation

For workstations, in addition to the NCSC B1 requirements, there is an additional set of security requirements defined by MITRE for the Defense Intelligence Agency [Woodward86]. The primary additional feature specified in that document is "floating" labels, which provide a mechanism for tracking the sensitivity label of all data which served as input to an object, such as all the processes which wrote into a file, or all the data written to a window. Sun is planning to build extensions into a future release of Secure SunOS to satisfy these requirements. These features will be provided on top of, not instead of, the basic B1 functions.

Administrative Interfaces

Closely related to the inherent presence of a network in the Secure SunOS configuration are the attendant problems of administering a widely distributed collection of individual machines. Sun will be developing administrative tools and interfaces to make this simpler, and also to allow subdivision of administrative roles and responsibilities.

Higher Criteria Levels

Sun has not yet decided how to approach the higher TCSEC levels, but will certainly do so as market requirements demand. Thus far, the focus of Secure SunOS has been on security features: mechanisms customers can actually use and experiment with, and incorporate into their own applications. Although Sun

considers the Secure SunOS TCB to have a sound internal architecture, it is quite large, and was not engineered to meet the B2 (or B3) requirements. Rather than building a brand new B2/B3 system now, however, Sun plans to wait and gain more experience with the new features provided by Secure SunOS, to know what to keep and what to leave out.

MAJOR FEATURES OF Secure SunOS

All UNIX system-derived secure systems face a similar set of design decisions for implementing security in the kernel: how to label files, how to handle directories, what to do about interprocess communication, etc. For the most part, these issues are addressed in Secure SunOS the same way they have been in other "secure UNIX" systems, such as IBM's Secure Xenix, the LINUS IV prototype developed at MITRE, etc. Like those systems, Secure SunOS follows the same basic model as Multics in applying mandatory security:

- Processes, files, directories, and all other objects have labels, and in the initial release, labels support 256 hierarchical mandatory security levels and 64 non-hierarchical mandatory security categories.
- The formal security model for the system is the Unified Multics version of the Bell and La Padula model described in [Bell76].
- For most objects, the model is restrictively interpreted to allow read-downs (reading data with a lower label than the process), but not write-ups (modifying data with a higher label than the process). This is done to simplify the implementation and eliminate a variety of covert channels at the design stage.
- Labels in the file system hierarchy are monotonically non-decreasing; all objects in a directory have the same label as the directory itself, except for directories, which may be "upgraded" (have a higher label than their containing directory).
- The initial version of Secure SunOS does not include any "least privilege" mechanism to replace use of *root* as the only form of privilege. Sun is, however, working with the /usr/group standards committee to define such a mechanism for eventual inclusion in the POSIX standard and future versions of Secure SunOS.
- The initial version of Secure SunOS also does not include any form of Access Control Lists (ACLs), and again, Sun is working with the standards committee to define an ACL interface for POSIX and future Secure SunOS systems.

The remainder of this paper describes some of the Secure SunOS features that are unusual and how they are different from other "secure UNIX" systems.

Labeled Windows

Probably the most important feature of Secure SunOS is that the on-screen windows are considered objects by the TCB, and have labels. This allows a user at a workstation to view, simultaneously, activity of processes with several different labels. The user can interact with these processes simply by moving the mouse from one window to another, unlike conventional mandatory access control systems, where often a logout/login

sequence is required every time labels are changed. Data can even be moved between windows of differing security labels, provided that the mandatory access control rules are followed.

A user interacts with the system by logging in at some level and creating windows of that level and below.

Auditing

Auditing in Secure SunOS is done on the basis of event classes. Each process has an audit state, specifying which event classes are audited for that process. The audit state specifies separately whether to audit a particular class for successful operations or failed attempts, so that, for instance, all access denials will be audited, but only successful write accesses (not reads) will be.

There is a system default set of audit classes, and each user has two sets that modify the system default. The administrator may specify, on a per-user basis, which event classes are always audited for the user, regardless of system defaults, and which event classes are never audited.

At present, Secure SunOS defines 13 audit classes, which include operations such as *data-read* (open for read, file status inquiry, etc.), *data-write* (open for write, etc.), *data-access-change* (change file permissions, change owner, etc.), *data-create* (create, delete, link, etc.), *login* (interactive login, logout, use of *su*, process created by *at*, etc.), and others. Audit messages are also generated for all administrative and privileged activity, identifying the specific operation performed as far as possible. Administrative events are audited specifically, not just as the administrator's access to a file; for instance, use of *vipw* to change the *passwd* file (where user registrations are kept) audits the change between the original *passwd* file and the modified version.

Auditing in a Network Environment

Because many of the machines in a Secure SunOS configuration are typically diskless workstations, audit messages are written across the network. Each machine has its own *audit daemon*, which collects the audit messages generated locally and writes them out to an audit file. The audit file is written using normal file I/O, but by using the Network File System (NFS) it is possible for the audit file to be located on an arbitrary machine anywhere in the network.

If an audit file fills up, or if the machine containing it goes down, each local audit daemon that was using that file detects the error and tries to create another audit file in a different directory. Because each local audit daemon has a list of directories to try when creating audit files, this is a very robust system: it is very likely that audit messages will find a home, even though they may be scattered among many machines.

Each machine in the system is responsible for auditing its own activity. This is safe because the machines are required to be physically secure, and because the non-privileged users do not have the capability of logging in as *root* or bringing the machines up in single-user mode. Thus, access to a machine does not present the opportunity to breach even that machine's own security.

Audit Reduction and Display

Because multiple audit files are an inherent feature of auditing in a distributed environment, Secure SunOS treats this as a feature,

rather than an inconvenience. The administrator wishing to view the audit trail uses the NFS to access all the audit files that are scattered among directories on machines all over the system, and the audit tool puts those records back in order.

The audit display tool is normally used to display the output of another program, *auditreduce*, which combines the audit records from many audit files and writes them out in time-sorted order for printing or report generation. Selection of audit messages (by type, string match, time of day, user name, security level, etc.) is performed in *auditreduce*, so that a report generation program need not have any selection options of its own. These tools also make it easy for the administrator to gather multiple audit files back into one place, keep old audit files on tape, and even keep selections from audit files online (such as a file containing only login records, but from the last six months).

Hidden Subdirectories

Every "Secure UNIX system" faces the problem of dealing with shared directories for temporary files. In Secure SunOS, as in IBM's Workstation Xenix, the temporary directories are special in that normal references to them actually translate to references to a subdirectory, of which there is one for each different label. A directory which causes this indirection (such as */tmp*) is known as a "hiding" directory, and its subdirectories (one for each security label) are known as "hidden subdirectories". These hidden subdirectories are created dynamically the first time a process tries to reference one that doesn't exist. Because they are created automatically, it is safe to delete them (as long as they are empty) at any time. Because use of hidden subdirectories in Secure SunOS is controlled on a per-directory basis, this facility is used not only for */tmp*, but also for */usr/tmp*, some of the directories in */usr/spool*, etc.

Physical Security

Every machine in a Secure SunOS configuration must be physically secure. What this means varies from customer to customer, but Sun can provide a variety of protective packagings that can make it arbitrarily difficult for an unauthorized user to breach the hardware integrity of a machine. Modified PROMs prevent the machines from being booted in single-user mode or from other than the default version of the kernel without a special (per-machine) password. The per-machine password may be changed by a system administrator.

REFERENCES

- [ATT86]
System V Interface Definition, Volumes I and II, 307-127, AT&T, Indianapolis, Indiana, 1986.
- [Bell76]
Bell, D. E., and La Padula, L. J., *Secure Computer Systems: Unified Exposition and Multics Interpretation*, MTR-2997 Rev. 1, MITRE Corporation, Bedford, Massachusetts, March 1976.
- [DOD85]
Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, Department of Defense, Washington, D.C., December 1985.
- [Woodward86]
Woodward, John P. L., *Security Requirements for System High and Compartmented Mode Workstations*, MTR 9992, MITRE Corporation, Bedford, Massachusetts, May 1986

TAXONOMY OF COMPUTER VIRUS DEFENSE MECHANISMS

Catherine L. Young

Office of Research and Development
National Computer Security Center
9800 Savage Road
Fort George G. Meade, Maryland 20755-6000

ABSTRACT

Computer virus defenses can be categorized using the following six grouping schemes:

Appearance versus Behavior,
Prevention versus Detection,
Executable versus Source,
Required Protection,
Performance, and
Ease to Implement.

Each scheme is explained, and examples are used to clarify each scheme. This taxonomy will aid in evaluating virus defenses and provide a foundation for designing new virus defenses.

INTRODUCTION

A computer virus is a piece of harmful code that is hidden in an otherwise normal program. A virus is also able to write a copy of itself to (or "infect") other programs.[5] This capability makes a virus especially dangerous to computer systems because a virus is likely to be harder to remove and is likely to have access to more computer resources than malicious code that does not have this capability. A virus could propagate for a certain period of time until some event triggers it to perform its harmful action. Because of the danger that viruses pose to computer systems, it is important to develop defenses against them.

The purpose of this paper is to categorize computer virus defense mechanisms. By presenting virus defenses in an organized manner, this paper should help virus researchers find defense categories that might be missing and to formulate more defenses for some of the categories. Six different schemes are described for classifying computer virus defenses and examples are given to clarify each scheme:

Appearance versus Behavior,
Prevention versus Detection,
Executable versus Source,
Required Protection,
Performance, and
Ease to Implement.

The first part of this paper describes four defense measures that will be used as examples. The bulk of this paper is the taxonomy which looks at the six categorization schemes. The paper explains each scheme and describes how and why each example fits into the various categories. Appendix A contains a matrix that shows a wide range of defense measures and how they are cataloged. Also included in Appendix A is a list of short descriptions of the defense measures. Appendix B consists of definitions.

EXAMPLE VIRUS DEFENSES

The following are four examples of virus defenses that will be used to clarify the taxonomy. These examples are a subset of the virus defenses described in Appendix A.

1. Coding Style Analyzer

A coding style analyzer uses the structure and content of a program to determine how many different programmers contributed to the program and what sections of code were written by each. A coding style analysis is related to the analysis of such things as handwriting and structure of sentences to authenticate the author of a book. An example of some coding style indicators at the source code level might be the number of spaces used to indent a While Loop, the frequency of comments, and the kinds of instructions used. Such an analyzer can be used to detect the presence of a virus because most virus code will have a different coding style than the host program, and the virus programmer's style is likely to be present in all the infected programs.

2. Prefix and Postfix Checker

Primitive viruses will probably reside in the beginning or end of host programs and will infect host programs with exact replicas of themselves. A prefix and postfix checker compares the beginning and end of files to see if a group of files have the same beginning or the same end. A group of files with identical prefixes or postfixes are likely to contain a virus.

3. ROM Devices

Programs may be put into read only memory (ROM) devices to prevent viruses from infecting critical programs. Programs stored in such devices cannot be modified. Adequate measures must be taken to make sure that only uninfected programs are stored in the ROM.

4. Intrusion Detector

An intrusion detector is a defense measure that monitors the activities in a system to determine if it is under attack. The detector does this by comparing current actions with past actions to see if something out of the ordinary is occurring. This defense can be used to detect many different types of intrusion, including viruses. Some distinguishing actions that would indicate a possible virus attack would be accessing many files, searching directories, and writing to executable files.[4]

TAXONOMY

A. Appearance Versus Behavior

The appearance-versus-behavior categorization distinguishes between virus defenses that detect or prevent infection by program appearance and those that detect or prevent infection by program behavior. An appearance defense considers the contents of a program, whereas a behavior defense considers the actions of a program. Appearance defenses work before the host program is executed, and behavior defenses work during execution.

Both kinds of defenses have their limitations. For a given program it may not be possible to absolutely determine by its appearance if that program contains a virus. These defenses make an educated guess as to whether a program contains a virus. For every defense, however, a clever virus can probably be written to outsmart that defense. The behavior defenses in many cases can be thwarted. A clever virus would behave in a subtle way so that its actions will not seem out of the ordinary for the host in which it resides.

The coding style analyzer is an example of an appearance defense because it looks at the contents and structure of a program to determine the author(s) of the program. The prefix and postfix checker also fits in the appearance category. This checker looks at the beginning and end contents of files to find identical prefixes and postfixes.

ROM devices qualify for the behavior category since they prevent viruses from performing the action of writing to the files stored in ROM. The intrusion detector is another behavior defense because it monitors actions.

B. Prevention Versus Detection

All virus defense measures fall into one of two categories: prevention or detection. Prevention defenses are those that stop virus propagation. Some of the actions involved in propagation that can be prevented are writing to executable files and accessing a file that has been touched by too many processes. Prevention defenses will control and limit access to files. Detection defenses recognize virus attacks but do not

have the power to stop a virus. Detection measures could be considered a type of prevention, however, because the system security officer can shut down the system to prevent further damage once the virus has been detected.

Prevention defenses, by definition, involve restricting an action and will only involve behavior measures. ROM devices will prevent virus spread because they are designed such that the information stored on them cannot be altered.

Detection defenses are usually appearance measures. A coding style analyzer detects the presence of a virus after the host program has been infected. It will indicate an additional programmer than the one(s) who wrote the host program. The prefix and postfix checker will detect a virus that resides in the beginning or end of files after it has spread to several files. The intrusion detector watches for telltale actions.

C. Source and Executable

Virus defenses can be partitioned based upon whether they monitor source code or executable code. Many of the proposed defenses listed in Appendix A are effective for both source and executable code. Since viruses can reside and propagate to either kinds of code, it is important to defend against viruses in both kinds of code.

A coding style analyzer works best on source code because source code will bear all the marks of its author. Executable code, on the other hand, is the result of a compiler converting source code into a standard form that a machine can understand. The resulting executable code will not have all of the marks found in source code such as indentations and comments. The more transformation processes that a program goes through, the less marks it will have of the author.

The prefix and postfix checker will work well with source and executable code. If a virus propagates by writing exact copies of itself to the beginning or end of programs, making the bytes exactly alike, the prefix and postfix checker will detect this regardless of the type of code.

Since the information in a ROM device cannot be changed, a ROM is used to store the final executable version of programs. For the most part, source code only needs to reside in the development system, and not in the target system. It would usually not make sense to store source code on a ROM device. A ROM device is, therefore, considered a defense for executable code.

The intrusion detector concerns itself with the actions. This defense falls under the executable category because source code does not act but executable code does.

D. Required Protection

Virus defense mechanisms must be

protected. The protection needed for each defense gives another way to classify virus defenses. The categories are based on what needs protection, and what type of protection is needed. The number of different protections and the kind of protection needed will have an impact on the vulnerability of a defense measure. Except for the ROM devices, all the measures included in this paper must be read-, write-, and execute-protected.

Each defense must be read-protected so that viruses cannot learn the threshold values and other information that would help them evade the defense. Write-protection is necessary so that the defense cannot be modified to ignore or help a virus. Each defense must be execute-protected to insure that it is called when it is supposed to be called and that the virus does not execute the defense in order to see if it would pass or fail. These three protection needs will not be enumerated for each defense.

The coding style analyzer requires protection of data that it stores temporarily while it is running. The file attributes that the defense analyzes must be read-protected so that a virus will not know what attributes are being analyzed.

The prefix and postfix checker also needs read-protection of the size and extent of its search and comparisons.

A ROM requires the physical protection of the computer to prevent an attacker from replacing the ROM with a virus-infected ROM. Once a program has been written to a ROM device, it cannot become infected.

The intrusion detector requires read- and write- protection of long-term and temporary data. The long-term data that needs protection is the audit records that it uses to determine the expected behavior of the system. The temporary data that must be protected is the expected behavior patterns which frequently change.

E. Performance

The effect that a defense measure has on the performance of a system gives another grouping of the defenses. This paper considers the following aspects of performance: the time it takes for the defense to execute, the amount of primary memory required by the defense program, the frequency of use, and the affect of the defense on the throughput of individual programs. The affect on throughput is a function of the other aspects of performance.

The coding style analyzer will be complex and will, therefore, take a relatively long time to execute. The coding style analyzer will take longer to run than the prefix and postfix checker because of its complexity. The difference in run time because of complexity will be multiplied by the number of files that need to be checked. This analyzer will require a moderate amount of memory for its code and the file attributes it must analyze. After it checks one file, the coding style analyzer will not

need the data generated for that file, so the corresponding memory can be reused. The coding style analyzer can be a background process that is executed on demand. If executed in the background, it should have small impact on the throughput of individual programs.

The prefix and postfix checker will be a simple defense that takes a relatively short time to execute. It will also take a small amount of memory for its code and data. The prefix and postfix checker may need to retain information from the files it checks, such as all the prefixes that are contained in more than one file (or some threshold number of files) and lists of which files contain each prefix. Even so, the amount of information to be stored will be small. The prefix and postfix checker can also be a background process which is executed on demand. Because of its simplicity, this defense will have a very small impact on the throughput of individual programs.

Time of execution, amount of memory required, and frequency of use do not apply when considering ROM devices because they are not software defenses. The main consideration is the affect that ROM's have on the execution of individual programs. The affect of using ROMs will vary depending on the system and the devices used. Some ROM's will be slower than the corresponding random access memory (RAM) and some will be faster. In the case of an IBM PC or XT, a ROM would improve performance because its programs do not need to be loaded off of a disk as do programs using RAM.

The intrusion detector will be a complex defense that will run continuously in the background, keeping track of what is going on in the system to determine if the system is under attack. The amount of memory required will be rather large, but it will vary depending on the sophistication of the detector. This defense will increase the execution time of individual programs more than the other examples due to its complexity and because it will be continually monitoring the system.

F. Ease To Implement

When evaluating virus defenses, one must consider how easy it will be to implement each virus defense measure with respect to current technology. We can classify defenses as being (1) easy to implement with current technology, (2) possible with current technology, (3) requiring much work to implement, and (4) requiring innovative ideas.

ROM devices are easy to implement and are widely used already. The prefix and postfix checker would also be easy to implement with current technology, using a simple comparison program. While it is possible to implement the coding style analyzer with current technology, it is not a trivial task. The intrusion detector, however, will require innovative ideas because it is not well defined which actions indicate that a system is under a virus attack.

CONCLUSION

This taxonomy should help those who are researching and developing virus defenses. They can use this tool in choosing and evaluating existing virus defenses. A systematic approach should help prevent researchers from overlooking significant characteristics of virus defenses. This organization of virus defense measures provides a foundation for designing new virus defenses.

APPENDIX A

VIRUS DEFENSE MATRIX

This appendix shows a wide range of defense measures and how they would fit into the taxonomy. The coding style analyzer, the prefix and postfix checker, ROM devices, and the intrusion detector defenses were chosen as examples for this paper. The column headings are the categories of the taxonomy and the row headings are the defense measures. The values in each column are explained in the legend that immediately follows the defense matrix.

Virus Defense Matrix

	A	B	P	D	S	E	R	P	P	P	E	I
	pe	re	ox	er	e	am						
	ph	et	ue	qo	r	sp						
	ea	ver	c	ut	f	el						
	av	ec	cu	ie	o	el						
	r	in	t	et	rc	r	tm					
	a	o	t	/	a	e	m	o	e	n		
	n	r	/	b	d	i	a	n				
	c			l	e	n	c					
	e			e	n	c						
	/											
I. Attribute of Change												
A. Message of Modification	b	d	e&s			t1 m1	1					
B. Second Copy & Compare	a	d	e&s	dw	tw	t2 m1	1					
C. Selected Portions of Programs	a	d	e&s	dr	dw	t1 m2	1					
D. Length of Program	a	d	e&s	dr	dw	fd i1	1					
E. Date/Time Stamp	a	d	e&s	dw		t1 m1	1					
F. Checksum	a	d	e&s	dr	dw	fd i1	1					
G. Encryption	a	d	e&s	dr	dw	t3 m2	1					

II. Virus finders

A. Prefix & Postfix Checker	a	d	e&s	tw		t2 m2	1					
B. Pattern Matcher	a	d	e&s	tw		fd i1	2					
C. Coding Style Analyzer	a	d	s	tr	tw	t3 m3	3					
D. "Antibiotic" Program	a	d	e&s	tr	tw	fd i2	4					

III. Execution Limitations

A. Access Control Mechanisms	b	p	e	dw		t2 m3	1					
B. Limited Domains	b	p	e			fe i2						
1. User-definable Domains	b	p	e	dw		t2 m3	4					
2. Type/Domain	b	p	e	dw		fe i2	3					
C. Low-level Minus One	b	p	e	dw		t2 m3	2					
D. ROM Devices	b	p	e	mp		fc i0	1					
E. Flow Distance	b	p	e	dr	dw	t2 m4	3					
F. Flow Lists	b	p	e	dr	dw	fe i4	3					
G. Intrusion Detector	b	d	e	dr	dw	t5 m4	4					

Legend For Matrix

Appearance/Behavior

- a = Appearance
- b = Behavior

Prevention/Detection

- p = Prevention
- d = Detection

Source/Executable

- s = Source
- e = Executable

Required Protection

This column will include pairs, with one item from each of these two groups:

Item:

- m = Machine
- d = Data objects (long-term)
- t = Temporary working storage

Type of Protection:

- r = Read
- w = Write
- p = Physical

Performance

This column rates four areas that affect performance. The first element of each item is a letter representing one of these areas as follows:

- t = Time required to execute the defense program once.
- m = Amount of primary memory used for defense program code and data.
- f = Frequency of execution of the defense.
- i = Affect on throughput of individual program execution.

Numbers are associated with the letters t, s, and i in this column, and they give a relative measure of the affect each defense will have on each aspect of performance of the system, on a scale of 0 to 5. The larger the number in the column, the greater the impact on that aspect of performance.

The frequency of execution is handled separately. The letters paired with the f for frequency are as follows:

- i = Interrupt driven; only called when a suspicious event occurs.
- d = On Demand; done in the background.
- c = Continuously in the background.
- e = When protected programs are executed.

Ease to Implement

The numbers in this column specify how easy it will be to implement each virus defense measure with respect to current technology.

- 1 = Easy to implement with current technology.
- 2 = Possible with current technology.
- 3 = Requiring much work to implement.
- 4 = Requiring innovative ideas.

Brief Description of Virus Defenses

I. ATTRIBUTES OF CHANGE

These defense measures monitor programs to see if they have been modified. The appropriate authority should designate which files to monitor.

1. Message of Modification - This defense is a modification to an operating system that sends a message to the System Security Officer's screen, or to an audit file, whenever a protected file is modified.

2. Second Copy & Compare - This defense can be divided into two parts: installation and comparison. The installation involves storing the second copy of all the files that are to be monitored. The comparison compares the current copy and the stored second copy to see if any files have been modified. The comparison can be done as a background process on demand.

3. Selected Portions of Programs - This defense is similar to the second copy and compare defense and can be divided into installation and comparison. This defense installs files to be monitored by using an algorithm to select portions of these programs and then storing these selected portions. To check to see if the programs have been modified, the defense will apply the algorithm again and make sure that it gives the same selected portions that it stored earlier. This comparison can be done as a background process on demand.

4. Length of Program - This defense computes and stores the length of the files to be monitored. It checks for modification by recomputing the length and comparing it to the stored length.

5. Date/Time Stamp - This defense stores the date and time that each monitored file was last officially modified. Each monitored file will have a date/time stamp in its header that indicates the last time it was modified. The defense mechanism will check to make sure the stored time and date match the stamp in the header of each file. This defense measure assumes that the system will protect the headers of files from being modified by any program that is not authorized to do so. The virus may have access that allows it to modify a program,

but most systems put a greater restriction on who is allowed to modify a header to a program.

6. Checksum - This defense computes and stores a checksum for each file that is monitored. To check to see if the files have been modified, the defense will recompute the checksum for each file and compare it with the stored checksum.

7. Encryption - This defense encrypts the files that are to be protected. Prior to execution these files will be decrypted. If a virus tries to write to an encrypted file, the result will be garbage.

II. VIRUS FINDERS

These defenses can recognize a virus by its appearance.

1. Prefix and Postfix Checker - This defense compares the beginning and end of files to see if a group of files have the same beginning or the same end. A group of files with identical prefixes or postfixes are likely to contain a virus.

2. Pattern Matcher - The pattern matcher will look for matching byte patterns in groups of files. Since viruses are likely to propagate by writing copies of themselves, identical byte patterns may indicate the presence of a virus.

3. Coding Style Analyzer - A coding style analyzer determines the distinctive techniques used by each person who contributed to a program. If the analyzer indicates that a given program has more programmers that it should have, this may be the result of a virus. If several otherwise unrelated programs have the same programmer for part of their code, this could also indicate the presence of a virus.

4. "Antibiotic" Program - This defense will recognize a virus based on the types of instructions it will need for propagation, triggering, and performing its mission.

III. EXECUTION LIMITATIONS

These defenses will prevent or detect viruses during program execution.

1. Access Control Mechanisms - Access control mechanisms allow users to decide who is allowed to access each of their files and what kind of access they will allow to others. A virus can only use the accesses of its host program. Access control mechanisms can slow viruses but will not stop most viruses.[3]

2. Limited Domains - This type of defense limits the objects that each user has access to.

a. User-definable Domains - With this measure, each user decides what objects he will need to access. He will restrict himself to only that set of objects (called a domain) which he will need.[6]

b. Type/Domain - For this defense, a domain is a group of programs. To access a given type of object, a subject must be part of a domain that is allowed access to that type of object.[2]

3. Low-level Minus One - This defense uses the Bell and LaPadula model.[1] To protect a set of executable programs, assign them to a level that is below the lowest level of all the other objects in the system. If the simple security property and the *-property are enforced, these low-level minus one programs could be read by all subjects, but they could not be written to by any subjects at the higher levels.

4. ROM Devices - Another way to protect programs from infection is to store them in ROM's.

5. Flow Distance - The flow distance defense will limit how far information can flow in a system. Each file in the system is assigned a flow distance based on how far the data in that file has travelled. For example, if a user writes a program that does not accept any input data, then that program has flow distance of zero while it is in the originator's possession. If the programmer gives this program to someone else, then its flow distance is increased by 1. The flow distance of a program that accepts input data will be increased by the flow distance of the input data. The system can limit the spread of viruses by restricting the flow distance of data. If a file receives a flow distance that exceeds the maximum, then that file can no longer be used.[3]

6. Flow List - The flow list defense maintains a list of each object that indicates which users have accessed that object. The system can then prevent an object from being accessed if too many users have accessed the object or if some undesirable combination of users have accessed the object. This defense could also be used on the individual level so that a user would only accept objects whose flow list contains only users that he trusts.[3]

7. Intrusion Detector - This defense looks for suspicious behavior that might be indicative of a virus or some other intrusion.

Propagation - A computer virus propagates or infects by writing a copy of itself in another program when the virus is executed.[5]

BIBLIOGRAPHY

- [1] Bell, D. E., and L. J. LaPadula. "Secure Computer Systems: Unified Exposition and Multics Interpretations." Technical Report MTR-2997, MITRE Corp. Bedford, MA. July 1975.
- [2] Boebert, W. E., and R. Y. Kain. "A Practical Alternative to Integrity Policies." Proceedings of the 8th National Computer Security Conference. Sep 1985.
- [3] Cohen, Fred. "Computer Viruses: Theory and Experiments." Proceedings of the 7th DoD/NBS Computer Security Conference. Sep 1984.
- [4] Denning, Dorothy. "An Intrusion-Detection Model." Proceedings of the 1986 IEEE Symposium on Security and Privacy. Apr 7-9, 1986.
- [5] Discussions with and reading of informal papers by Joseph Beckman, O. Sami Saydjari, and Timothy Kremann of the National Computer Security Center.
- [6] Smith, Terry A. "User Definable Domains as a Mechanism for Implementing the Least Privilege Principle." Proceedings of the 9th National Computer Security Conference, Sep 1986.

APPENDIX B

DEFINITIONS

Computer Virus - A computer virus is code that resides in a program that can copy itself onto other programs. Computer viruses have the potential to do great damage to computer systems by propagating and later performing devious acts such as deleting files.[3, 5]

Host Program - A program that contains a virus.[5]

Computer Viruses: Myth or Reality ?

Howard Israel
National Computer Security Center
9800 Savage Rd.
Fort George G. Meade, MD 20755-6000

Abstract

This paper will show that a computer virus [COHEN] may be no more a threat to computer systems than a Trojan Horse and any protection mechanism that will work against a Trojan Horse will also work against a computer virus, specifically a mandatory policy (e.g., [BELL/LAP] [BIBA]). In addition, it will discuss two possible protection mechanisms that address the Trojan Horse threat.

Background

A computer virus is a program that propagates itself [COHEN]. Depending upon its design, a virus may propagate itself on a limited basis or more extensively through the file system. That is, it may selectively propagate itself so that only one copy exists at any one time in the system [THOMPSON], it may slowly spread through the system, or it may propagate as fast and as often as possible in the system.

A virus may act as a Trojan Horse [ANDERSON] (hereafter referred to as a 'viritic Trojan Horse') by performing an overt action (the advertised purpose of the code that the executor expects to occur), a covert action (typically benefiting the author and harming the executor of the Trojan Horse, which the executor does not expect to occur) and then propagate itself to other areas in the file system taking advantage of the executor's privileges and rights. Because a viritic Trojan Horse can 'flow through' the system (via the viritic feature) it may increase the likelihood of execution and number of executions.

D. J. Edwards identified the Trojan Horse attack in [ANDERSON]. In [KARGER], the concept of a Trojan Horse propagating itself was discussed, although there was no distinction made between a Trojan Horse that was viritic or not. The ARPANET collapse on October 27, 1980 was attributed to the accidental propagation of a virus [NEUMANN]. There are even references to viruses in modern science-fiction novels [BRUNNER].

Part I: Comments on Recent Research

1. Measuring Infection Times

To show that a viritic Trojan Horse was a significant threat beyond a non-viritic Trojan Horse, it would be necessary to compare the infection time [COHEN] of a viritic Trojan Horse against a comparable

non-viritic Trojan Horse. The use of a control group should adequately show if the viritic attribute will have an additional significant affect on the Trojan Horse threat.

This author welcomes any research in this area. For, if done properly, it will show a viritic Trojan Horse to be either a more serious threat than a non-viritic Trojan Horse or of no greater consequence. However, highly variable factors that will change over the life of the experiment include "the enticement" (this includes the advertised overt capability of the program as well as the methods used to "sell" it to the target user community) to execute the Trojan Horse and the knowledge of the user community, as well as other sorted variables such as: user activity level, time of day, etc. Researchers doing work in this area should be scrutinized fully when presenting results because of these "field" variables. Therefore, experiments must be designed and executed very carefully before any results should be considered credible.

2. Virus Affects on Systems with Fundamental Flaws in Security Policies

[COHEN] discusses the virus experiments that show "fundamental flaws in security policies". Any fundamental flaw found in a security policy need not necessarily use a virus to display the weakness. A non-viritic Trojan Horse should succeed in demonstrating any weakness sufficiently. There is no perceived advantage in using a viritic Trojan Horse (as opposed to a non-viritic Trojan Horse) to demonstrate a flaw in a security policy.

Although it may be easier, in some cases, to achieve a particular objective by using a viritic Trojan Horse, it has not been shown, nor does this author believe that it can be shown, that there is an objective a viritic Trojan Horse can achieve that a non-viritic Trojan Horse can not achieve on currently used computer systems.

It is also interesting to note that the experiments performed [COHEN] were executed on systems that either did not have an enforced mandatory "security policy" at all (i.e., UNIX, VM/370, VMS, Tops-20) or had only a partial implementation of a mandatory security policy (i.e., OS/1100 on the Univac 1108) [LEE], thereby, proving the obvious. The following discussion will describe the affects a Trojan Horse can have on a system

that enforces a mandatory policy.

Trojan Horse vs. a Mandatory Policy

The model described by [BELL/LAP] protects systems against unauthorized disclosure as defined in a specific policy. A Trojan Horse would have to take advantage of a covert channel to disclose information (in a properly implemented [BELL/LAP] system). The same holds true for a viritic Trojan Horse. Earlier work [COHEN] made the implication that the Univac 1108 fully implements the [BELL/LAP] model. This is not the case. OS/1100, as delivered by the vendor, has the concept of "security levels" and enforces the simple-security condition, but it does not enforce the *-property [LEE].

Note: a Trojan Horse whose purpose is to violate the integrity of a system [BIBA] could easily succeed in a system that only enforces the [BELL/LAP] model. Thus, it is always true that a system can only protect what it is designed to protect and not necessarily more.

A system that enforces an integrity model [BIBA] would protect against a Trojan Horse (viritic or not) that attempts to violate the integrity policy. In [COHEN] an erroneous conclusion that a system with both an integrity policy [BIBA] and security policy [BELL/LAP] must provide isolation was arrived. This would be true only if a single label were used for both the security and integrity policy enforcement (see Table D, below) [SCHELL]. One must consider the case described in Table C (i.e., that both policies may exist concurrently in a system without forming an isolation, or complete partition, between security levels [SCHELL]). The following simplified example illustrates this:

Assume:

"TS" and "U" are both clearances (on users) and labels (on objects) that enforce the security policy (i.e., read policy).

"H" and "L" are both clearances (on users) and labels (on objects) that enforce the integrity policy (i.e., write policy).

A "TS" labeled object is more sensitive to disclosure than a "U" labeled object. A "TS" cleared user (subject) is not permitted to write "TS" objects to a "U" cleared user (subject). A "U" cleared user (subject) is not permitted to read a "TS" object.

An "H" labeled object is more sensitive to modification and creation than an "L" labeled object. An "L" cleared user (subject) is not permitted to write an "H" object. An "H" cleared user (subject) is not permitted to read an "L" object.

Access modes:

R = Read
W = Write
Null = None

Permissible actions:

	Object		
	I	TS	U
Subject	TS	RW	R
	U	W	RW

Table A: Security policy (simplified).

As shown in Table A, the basic concern is to prevent an untrusted subject from reading sensitive objects. The flow of information tends to be from least sensitive to most sensitive ("U" to "TS").

Permissible actions:

	Object		
	I	H	L
Subject	H	RW	W
	L	R	RW

Table B: Integrity Policy (simplified).

In table B, the basic concern is to prevent an untrusted subject from writing (or creating) a high integrity object. The flow of information is from high integrity to low integrity ("H" to "L").

Permissible actions:

	Object				
	I	TS/H	TS/L	U/H	U/L
Subject	TS/H	RW	W	R	Null
	TS/L	R	RW	R	R
	U/H	W	W	RW	W
	U/L	Null	W	R	RW

Table C: Intersection of both a security and integrity policy.

Table C shows the relationship between security and integrity. It represents the intersection of the security and integrity policies defined above. A "U/L" subject can neither Read nor Write on a "TS/H" object. A "TS/H" subject can neither Read nor Write on a "U/L" object. These are desirable features, for they will stop the flow of a viritic Trojan Horse from one partition to the next, while still permitting the controlled sharing of information.

Permissible actions:

	Object		
		TS	U
Subject	TS	RW	Null
	U	Null	RW

Table D: Subject/object relationship when the same label is used for both "security" decisions and "integrity" decisions.

Table D shows the permissible actions that can occur on a system where the same label is used for both security and integrity decisions. The result is isolation between the two classes of users.

Summary

An enforced disclosure and integrity policy can provide an effective means of stopping several classes of Trojan Horse (both viritic and non-viritic) attacks, provided the mechanisms are defined in consideration of each other. These policies will not have an affect on attacks that invoke Denial-of-Service problems on a system, as the disclosure and integrity policies mentioned do not address Denial-of-Service issues.

While the above simplified example demonstrates the correctness of the approach, by allowing one category to be added to both the security and integrity labels each, the complexity of the access matrix increases to 256 different access cases (16x16). Although this may appear overwhelming, the defined policies can still be easily enforced, no matter how many levels and how large the category sets are defined for both the security and integrity policies.

There are systems available today that enforce a mandatory policy [MULTICS] [SCOMP]. These systems will be able to provide protection against Trojan Horse (viritic or not) attacks that attempt to violate the enforced mandatory policy.

Part II: Possible Methods to Defeat Viritic Trojan Horses

1. Comparison Utility

Without considering the objective of the Trojan Horse, it appears much easier to detect the presence of a viritic Trojan Horse that has successfully propagated itself (i.e., more than one copy of the virus exists in the system), than it would a non-viritic Trojan Horse. This proposed detection method would use a comparison utility to show the use of similar code in different files. Any similar code discovered may or may not be for legitimate reasons.

Consider a file system that has "n" files. It would require:

$$\frac{n(n+1)}{2}$$

comparisons on the files to completely detect a successfully propagated Trojan Horse (i.e., viritic Trojan Horse). If during the comparison process, code is found common to two programs, they would then be considered suspect. It would be necessary to "review by hand" to confirm or deny the presence of the viritic Trojan Horse. The code review would point out whether the "common code" has a valid purpose. What is being detected are similarities in code that, in principle, should not exist. This method is independent of the function of the (viritic) Trojan Horse. That is, it does not matter what the purpose of the viritic Trojan Horse would be to detect its existence.

This method could not be used to detect a non-viritic Trojan Horse for obvious reasons (i.e., only one copy of the Trojan Horse may exist, not several, as is likely, but not necessary [THOMPSON] with a viritic Trojan Horse).

Given the above possible solution to detecting a viritic Trojan Horse, several details remain. Detection depends upon how good the comparison utility is. It also depends upon how well the viritic Trojan Horse succeeds in implanting its 'child' into innocuous programs.

For a viritic Trojan Horse to implant itself successfully, it would have to be implanted in such a way as to guarantee:

- a) that the target program would remain operative, and
- b) that the virus would be put into a location such that the (entire) viritic aspect would be guaranteed to be executed.

If either of the two preceding conditions were not met, the success of the viritic Trojan Horse would be jeopardized.

One way to defeat the above detection would be for the viritic Trojan Horse to propagate itself such that the child's "likeness" was not the same as the parent's "likeness" (i.e., the code appeared different enough such that the comparison utility could not detect the similarity). This is perceived as a difficult, although not impossible, problem.

2. Spawning an Untrusted Process

By enforcing the least-privileged concept on a process by process basis, it is possible to provide a safe environment to execute untrusted code (which may contain a Trojan Horse) [DOWNS].

When a process wants to execute "untrusted" code (which the executor suspects contains either a viritic or non-viritic Trojan Horse), the process could then spawn a child process, which would include any necessary data. As long as the child's process access rights are limited with respect to the parent's process access rights, the parent process (and all associated data files) would be safe. Of course, anything in the system that the child process can access is a potential victim to the Trojan Horse, including other information located in the child's process (e.g., data deemed necessary to execute the untrusted code) and the results of the executed program.

If one considers the child process to be temporary (i.e., for the life of execution of the untrusted program) and the user can terminate the program at will, then the user will be able to protect the information managed by the parent process, which is the goal of this exercise.

This can be considered analogous to what a system administrator can do today. If an administrator (with the appropriate system privileges) wanted to execute untrusted code (e.g., a game program) he could perform the following:

- a) set up an account, such that the new account had no access to the administrator's privileged account or access to anything but publicly readable files,
- b) login to the new unprivileged account,
- c) execute the game program
- d) delete the unprivileged account.

Of course, if the unprivileged account had write access to any file in the system, the untrusted code (e.g., game program) could propagate a viritic Trojan Horse into the unprotected file, and thus be subject to further execution.

This is in addition to the obvious risk of unintentional disclosure, modification, or deletion of any data given to the game program to accomplish its task. (This risk would be nonexistent if the untrusted code did not need any user supplied data.)

The remaining problem with this scenario, then, is that the untrusted code could invoke a Denial-of-Service attack on both the user's process and the system. Since a well accepted model is lacking in this area, no solution is proposed.

Conclusion

A viritic Trojan Horse (i.e., computer virus) presents no new threat to computer systems. If the Trojan Horse problem were solved (for any class of Trojan Horse problems), the viritic Trojan Horse problem would also be solved (for that same class). Any solution to the Trojan Horse problem would also be a solution to the viritic Trojan Horse problem.

A security policy and an integrity policy (used in conjunction, in an intelligent manner) provide a reasonable protection scheme against Trojan Horse (either viritic or not) attacks. A Trojan Horse (viritic or not) may still invoke a Denial-of-Service problem, unless a model addressing this issue can be stated and enforced in a system.

While a viritic Trojan Horse is interesting, in the fact that it presents many novel attacks, it is no more dangerous than a non-viritic Trojan Horse attack. The viritic aspect of a Trojan Horse appears to be more of a red-herring, in the sense that it has taken attention from the basic problem.

Two partial solutions have been discussed. Each must be explored and experimented with in more detail. Better solutions for more classes of Trojan Horse attacks need to be advanced.

Acknowledgments

I would like to thank A. Arsenault, S. LaFountain, R. Morris, and G. Wagner for their insightful comments on early drafts of this paper. I would also like to thank J. Beckman and O. Saydjari for participating in interesting conversations on this topic. Thanks also go to S. O'Brien, C. Schiffman and R. Winder for their careful review of this paper and A. Arsenault, D. Gary and M. Tinto for their continuous encouragement.

References

- [ANDERSON] Anderson, J.P. (Ed.), "Computer Security Technology Planning Study, James P. Anderson Co., ESD-TR-73-51, Vols. I and II, HQ Electronic Systems Division, Hanscom AFB, MA, October 1972.
- [BELL/LAP] Bell, David, and LaPadula, Leonard, "Secure Computer Systems: Mathematical Foundations", 1 March 1973, MTR 2547, Vol. I, The MITRE Corp., 1973.
- [BIBA] Biba, Ken, "Integrity Considerations for Secure Computer Systems", USAF Electronic Systems Division, ESD-TR-76-372, 1977.
- [BRUNNER] Brunner, John, "The Shockwave Rider", Harper & Row, Publishers, Inc., 1975.
- [COHEN] Cohen, Fred, "Computer Viruses: Theory and Experiments", 31 August 1984, 7th DOD/NBS Computer Security Conference, Gaithersburg MD, September 24-26, 1984.
- [DOWNS] Downs, Deborah D., Rub, Jerzy R., Kung, Kenneth C., Jordan, Carole S., "Issues in Discretionary Access Control", Proceedings of the 1985 Symposium on Security and Privacy, April 22-24, 1985, IEEE Computer Society Press.

[LEE] Lee, T. M. P., "Future Directions of Security for Sperry Series 1100 Computers", 7th DOD/NBS Computer Security Conference, Gaithersburg MD, September 24-26, 1984.

[KARGER] Karger, Paul, and Schell, Roger, "MULTICS Security Evaluation: Vulnerability Analysis", Electronic Systems Division, June 1974, AD/A-001 120.

[MULTICS] "Final Evaluation Report of Honeywell Multics MR11.0", National Computer Security Center, CSC-EPL-85/003, 1 June 1986.

[NEUMANN] Neumann, Peter G., (Ed.), "Forum Risks to the Public in Computer Systems", 17 August 1986, Vol 3, Issue 38.

[SCHELL] Schell, Roger, "Biba's Integrity levels and viruses", Computer Security Forum, Vol 3, Issue 12, 29 May 1984.

[SCOMP] "Final Evaluation Report of SCOMP, Secure Communications Processor, STOP Release 2.1", National Computer Security Center, CSC-EPL-85/001, 23 September 1985.

[THOMPSON] Thompson, Ken, "Reflections on Trusting Trust", Communications of the ACM, Vol 27, No. 8, August 1984.

What do you Feed a Trojan Horse?

Techniques to Solve Hacking Problems

Cliff Stoll

Lawrence Berkeley Laboratory

Berkeley, CA 94720

Revised 29 May 1987

Abstract:

Computer security is sometimes best served by corking up known holes in a system, and sometimes by tracking an intruder to the source. Techniques used to pursue the latter course include high speed network traces, operating system alarms, off-line monitoring, and traffic analysis. But technical methods are not enough. It's just as important to coordinate efforts with law enforcement agencies and other professional organizations, and to understand the constraints set on each organization. Persistent sleuthing can ultimately locate the source, but it may require considerable time and effort.

Introduction:

When you know a computer system is under attack, you're presented with a choice: should the drawbridge be raised and outside access cut off, or should the source of the attacks be determined?

This paper addresses what to do when you choose to track the penetration. Related topics, such as how to detect an attack, or how to protect a system, are largely ignored here, although all of these topics are intimately intertwined.

Once you decide to find the origin of the attacks, you must start a traceback effort. Occasionally, this will be easy, and the suspected intruder collared quickly. Usually, the intruder will have taken steps to conceal the pathway into your system (often using stolen resources to do so), and unwinding the connections may challenge your best efforts.

Tracebacks through digital and analog networks are theoretically straightforward -- after all, an outside attacker made a physical connection into your computer. In practice, however, unwinding a complex connection can be quite daunting, especially in a short time.

Making a traceback is essential for the prosecution of an attacker; it also teaches lessons in network connectivity, coordination between law-enforcement agencies, and the strengths and weaknesses of our interlinked digital networks.

Our problem, then, is to unwind the connections made by an unknown intruder, and ultimately determine who's in the system. This effort requires a thorough understanding of operating systems, networks, telephony, and digital communications. Familiarity with legal issues and law enforcement protocols will be helpful. Fitting together diverse clues, some of which are misleading, eventually may lead to an answer, although a prosecution may not necessarily follow.

Should you ignore the attack?

There are good reasons *not* to try to catch an attacker. You may subject your system to danger -- the intruder may gain sufficient privileges to delete or modify important files*. You risk the disclosure of sensitive information. It may prove to be a wild goose chase. The attacker might be illusory -- a figment of your operating system. As we shall see, unwinding a complex connectivity can become expensive, requiring coordinated efforts of several technicians. Prosecution may prove impossible, due to legal problems, or infeasible, due to political or economic factors. You may embarrass your organization by admitting that an outsider has access to your computers.

If you decide not to trace the source of an attack, there are several alternatives available. You may ignore the intrusion completely. You may close your doors to the attack, by changing passwords, tightening modem access, or strengthening your operating system. Or you may simply legitimize the activity.

* Perhaps more damaging than a massive deletion of files is the slight modification of files -- this may go undetected.

There are also many valid reasons to chase down an attacker. The intruder may be out to injure your organization, possibly for personal benefit. Lost resources can be recovered only by locating whoever took them. A criminal may be caught and prosecuted by means of a traceback. As a community service, tracebacks of illegal activities help make networks safer for everyone.

Network tracebacks can be a rewarding area of academic research. Trying to catch someone within a computer can lead you through problems in operating systems, networking and network topology, as well as digital and analog telecommunications. Such work also touches on technology law and the ways in which various organizations respond to novel problems.

Organizing your efforts

Once you've decided to trace an attacker, you'll need to organize your efforts. Early on, designate one person to serve as a single point of contact until the problem is solved. Since your staff will want to know what's happened, stop rumors by holding a meeting.

You'll need to warn staff members to be quiet about the investigation. If the word leaks out, not only will your work have been wasted, but a malicious intruder might damage your system. Law enforcement organizations won't help you if they believe you may leak investigative information. Unless your staff realizes the sensitivity of this matter, they may mention it on electronic bulletin boards, at conferences, or to colleagues.

Start a logbook, collecting and analyzing your evidence within it. Record all suspicious activities, along with their dates and times. Maintain a clear distinction between conclusions based on firm evidence and suspicions based on indirect evidence or assumptions. Summarize telephone calls and messages from other sites. Keep your logbook out of any computer accessible from the system under attack -- assume that the intruder is searching for it.

You will repeatedly explain what happened to a variety of people, many of whom won't understand computer jargon. For this purpose, prepare a summary of what happened, using layman's terms. Describe exactly what damage has been done; include loss of services, disclosure of information, and the costs to rebuild lost files; quantify this damage in dollars, if you can.

Keep records of the costs of the break-in, and your expenses in repairing it. This is needed to determine the level of the offense (felony vs. misdemeanor); it also indicates which agency will have jurisdiction over the case (local/state/federal). Since some expenses in tracking and solving the problem can be recovered through lawsuit, these records can be essential should the case go to trial.

Many legal issues come up in trying to track and prosecute a computer intruder. What privacy rights exist? Can you bring suit for damages against someone breaking into your computer? Can you recover for the costs of tracking? What constitutes a breakin? The laws and interpretations have changed in the past year, so visit a law library to learn about current statutes involving computer crime. This is especially helpful in communicating with law-enforcement people, who may not be aware of recent codes¹.

Early on, determine how deeply you are threatened. Does the attacker have system privileges? Have Trojan horses, logic bombs, or virus programs been created? Is there a danger to your system if you try to catch the attacker? Have you the resources to chase down the attacker? Set limits on how much time and effort you will commit to the task.

Who should you tell?

Soon after detecting an attack, you should spread the news to people who can help solve the problem and to people running other systems at risk. But limit the spread of this news! Certainly, inform your management and your funding agency. If you have evidence of attacks via other systems, inform trusted system managers of those sites *by telephone* (never send computer security messages by electronic mail!).

Several external organizations may be able to help you. Your local police are charged with the enforcement of local and state laws -- you should be in close contact with them. Federal investigation and enforcement efforts are coordinated through the FBI and the Secret Service; the US Department of Justice will handle the prosecution in these cases. Problems which involve the Milnet, Arpanet, or military computers can be referred to the Defense Communications Agency and the Air Force Office of Special Investigations.

You should contact the National Computer Security Center in Ft. Meade; while they perform the difficult task of trying to prevent these problems, they also keep track of attacks, and can coordinate efforts to understand and solve such problems. Additionally, be aware of the Institute for Computer Sciences within the National Bureau of Standards. Both of these agencies can advise you on security holes which your intruder has used.

When traces must be performed over digital networks, it's important to be in close touch with the appropriate network operations center. It's important to know beforehand whom to call: when confronted with an attack, it's difficult to reach the correct person quickly.

Throughout your contacts with these organizations, keep records of who you've spoken to, and what response you've received. In addition to reinforcing your own memory, these records are helpful in prodding agencies to take appropriate action.

Operating System Accounting

Fundamental to the detection and tracking of any unauthorized computer user is adequate resource accounting. Modern operating systems typically record resource usage, task names, times of login, and connection port. Often, this is the only information available to determine the extent of an intrusion³.

The quality of auditing data varies with operating system, and with the system manager's needs. With good accounting data, and reasonable summaries of activity, audit trails are easily constructed. Spotty accounting, with inaccurate clock times and missing records, prevent the detection of even the most gross violations.

Even if a computer does not recharge for usage, accounting records should be kept. Without these audit records, it's impossible to reconstruct what has happened in the past -- an essential part of tracing an attacker. From this, it follows that no individual should be able to disable accounting. The accounting records should, at minimum, include port number used to access the computer, task name executed, flags for attempted access to protected files, and start/end times.

Accounting records can also be used to identify the incoming port and speed. If network connection

(e.g. Milnet or Arpanet) is indicated, the originating host name may be included in the accounting records. If the accounting records show a serial port, then determine the baud rate of this port: generally, high speed connections are from on-site, and low speed connections are from off-site, usually through modems.

The login/logout times are used as timestamps to control searches into other accounting records. These times are compared to local area network connection times or compared with telephone company billing records. To simplify record keeping, save these dates and times in GMT, and keep your clocks accurate to a second (non-synchronized clocks confuse traces across several systems).

If no obvious damage is done, (perhaps only files have been read), a successful invasion of a computer may go undetected for a long time. Thus, detailed accounting records should be saved for at least a year. These records can be used to show how an invader succeeded in entering your system, and can point out accounts which have been poisoned by the attacker.

In some circumstances, standard accounting records may not be trustworthy. An invader may have disabled accounting or modified accounting records. Accounting may be incomplete for some nodes on your network. In any case, ambiguous clocks and sloppy record keeping will confuse the interpretation of audit trails.

Local Area Nets

Almost every large computer system, and many smaller ones, use local area networks to interconnect terminals, modems, computers, and other networks. Often these are referred to by the manufacturer's name (Micom, Develcon, Sytek, etc.). They usually introduce significant holes into the security of a system -- an ideal place to plant Trojan horses. Seldom are these LANs programmed by the systems staff, or considered as security problems⁴; they are usually set up by a communications group, and then little more than routine maintenance is performed. Few systems people pay attention to the connectivity they provide^{**}. A connection through a LAN may be impressively difficult to trace.

^{**} For example, some local area networks allow outside dial-in users to immediately dial out from a common modem pool, forcing the host to pay for long distance calls, and providing an excellent hiding place for hackers.

When an attack originates from the LAN, it's necessary to find the originating port. Usually, accounting data from the operating system will tell which computer port the activity came in from. Historical accounting data from the LAN controller is needed to determine from which port or network the attack originated. Some LAN controllers simply cannot provide this information -- avoid using such systems! When this audit information is available, it's important to collect and save the data for later analysis. Make certain that the connections are recorded along with related housekeeping information, such as baud rates, dates, and times. Since there will likely be hundreds or thousands of connections recorded every hour, an accurate time stamp is essential -- periodically calibrate this clock.

When a local ethernet is suspected of being in the line of the problem, it may be necessary to audit all of the connections on the ethernet. Likely, this will require time-domain reflectometry to physically locate all of the drops on the cable. Because of the potential of promiscuous-mode listening, ethernet problems must be taken seriously⁵.

Telephone Traces

Eventually, a telephone trace may be needed. For many reasons, telephone companies may appear to drag their heels before tracing a call. It's often technically difficult, requiring skilled technicians, and phone companies may say that they are worried about the risk of lawsuits. Such statements appear unfounded⁺; there is no liability when acting under a search warrant. Most telephone companies have departments which are expert in tracing telephone lines, and telephone traces are common procedures.

A telephone trace can be obtained through the appropriate police agency, who will contact the District Attorney for a search warrant. Affidavits and relevant evidence will be reviewed before the courts, and a warrant issued. The telephone company will probably need some advance warning before installing the necessary equipment, or placing technicians on standby. This calls for advance coordination between the computer site, the police, and the telephone technicians. Advance dry-runs will help iron out problems⁶.

⁺ Such a backward telephone trace, (called a "trap and trace") has been held not to violate privacy rights, and does not require a court order when performed on your own telephone line. See 18 U.S.C.A 3121 (b) (1) and (b) (3). Indeed, some telephone companies are now offering residential service that displays the originating telephone number while the dialed phone is ringing.

Depending on the mechanism used, telephone traces may take place in real-time or after the fact. In either case, it may be necessary to know the exact start time of the incoming call to the second. When tracing a call through multiple exchanges or through long distance exchanges, simultaneous coordination by several groups may be required, since traceback equipment is seldom integrated with the long distance billing system. For real-time traces, telephone technicians can recognize digital traffic carried by its characteristic sound on a line, and it's usually straightforward to be certain that a particular connection is the valid one.

Because of deregulation, interstate and cross-carrier telephone traces can be difficult. Despite this, many complex telephone traces can be done within a matter of minutes, provided that all organizations along the line are forewarned. Local and long distance telephone systems need automatic traces to pinpoint troubles in lines and switching gear, so the equipment and techniques exist.

After a successful telephone trace, a law-enforcement agency may set a pen-register on the telephone line. Such a device records the phone numbers of all out-dialed calls⁺⁺, and helps determine the extent of an individual's telephone contacts.

Digital Network Traces

Packet switched networks, such as the Internet, have information on the originating node written within the packet header block⁷. When the network links directly to the host computer, packet information may be recorded by the accounting program. Some networks convert the packets into serial data streams (such as RS-232), and send this stream to the host; in such cases the packet header information is unavailable to the host, but may be available at the X.25 interface.

Packet header information may be counterfeit, garbled, or missing. When this is suspected, a call should immediately be made to the network operations center. Such organizations can quickly unwind the linkages within their systems and trace the path of a connection. Such unwinding can only be done while the connection is active; Internet does not record connections for later analysis. Other networks, such as Telenet, Datapac, and Tymnet, do save records for billing purposes, and historical connections can be reconstructed, provided that accurate times are available for comparison.

⁺⁺ The installation of a pen-register and related recorders requires a court-order.

Digital networks are worldwide, and some information may be hidden when network boundaries are crossed. For this reason, international traces require close cooperation between the network operations people. Find out in advance who is responsible for these traces, and exchange telephone numbers. Know your network location and port number as referenced by the network -- it can save several minutes in performing a real-time network trace.

Digital networks which use datagram ack/nak flow control can be timed to determine round trip travel time. On the Milnet/Arpanet, many seconds may elapse before a datagram is acknowledged on a coast to coast connection. After accurately timing these packet receipts, a statistical average will indicate a nominal distance to the originating site. A similar method can be used when Kermit or Xmodem protocols are used over serial lines.

Some intruders will use each successive computer as a jumping off point to get into another system. After stringing together several computers, modems, and a variety of networks, such connections may become frustratingly complex. Leapfrogging between computers and networks can allow an intruder essentially toll-free access to many systems, with intermediate sites paying for the connections. Fortunately, interactive response time suffers, and these users eventually simplify their connectivity.

Alarms and Monitors

When an attack is suspected, it's important to determine exactly what information is being sent out of the system, and what files are being accessed by the intruder. Accounting information provides a pointer, but there is no substitute for a complete printout of the bidirectional chatter. This can be recorded by the operating system, but off-line monitoring is easier, better hidden and entails less overhead. Recording equipment (such as a PC with serial lines and a hard disk) can easily be daisy-chained[†] to modems, providing continual monitoring of all serial traffic. Network software (such as the TCP/IP daemons) are easily modified to save traffic, as well; this can be done on-line or off-line.

[†] RS-232 data lines can be multi-dropped to drive several receivers.

The monitoring software and equipment can be programmed to alarm whenever particular character sequences are detected. Thus, an alarm can be set off whenever a particular account is accessed or whenever a certain password is entered. You may wish to build more sophisticated alarms, using expert-systems techniques.

When immediate response is needed, alarms can be connected to an auto-dialer^{††}, to ring a telephone or pocket pager. Since many intrusions last for only a few minutes and may occur at any hour, a pocket pager is essential to quick tracing of these calls.

Operating system alarms usually warn the system manager when false logins have been detected, or when protected files have been read. These are very useful for the detection of intruders, but cannot be fully trusted. When an invader acquires system privileges, it's likely that he will disable accounting, turn off the alarms, and modify any files that record his presence. These on-line alarms, then, aren't very dependable once an attack has succeeded.

Traffic analysis

After monitors have recorded the intruder's traffic, analyze what has happened. Annotate and save all printouts; keep a detailed record of what happened in your logbook. Using the printouts, try to determine what the intruder was looking for, what he tried to access, and what keywords were important. Did he try to link to another computer? What passwords did he try? Did he modify any of your files? How long did each session last? What attracted his interest?

Of course, each intrusion will be different; detailed traffic analysis is crucial to the solution of the problem, partly because it describes the intruder's interests, and in part because it provides law-enforcement agencies with evidence useful in the prosecution of the intruder. Keep all of these records off line -- never allow your security related records to be readable from any computer network or dial-up line.

^{††} Hayes-compatible modems make excellent auto-dialers, and can be programmed to send information to pocket pagers.

Communications with other sites

When an attack on your site has been detected, everyone wants to know about it. It's necessary to strictly limit the spread of this information if you wish to trace the problem. To prevent rumors from spreading, hold meetings to discuss progress, warning all members that the information should not be spread. Talk openly with trusted site managers, but do not leak information to the press, or to various bulletin boards.

It's essential that all communications be kept out of electronic mail, and no files be kept anywhere of this activity. Intruders will naturally scan file systems, searching for keywords that might indicate that they have been detected. Electronic mail is an especially fruitful section to search.

When coordinating work with another site, communicate by telephone. Keep any files related to this activity encrypted. Assume that all of your files are regularly read by the intruders. Do not keep files with obvious names like "security" or "hacking". When building monitoring programs, do give away their function with titles like "monitor" or "watchdog".

Relationships with Law Enforcement Agencies

Presently, the federal government and several states have tight computer security laws. These are enforced through the FBI, the Secret Service, and various state and local police agencies. Police training and awareness has been recently increased, although most of it seems to be directed towards computer crimes with direct, measurable economic implications (e.g. theft via computer, accessing bank records).

For a poorly researched case, there isn't much hope for enforcement due to the novelty of the laws, the lack of judicial caselaw, and the need for highly trained specialists. A well documented case, including a detailed analysis of the losses, will increase the chances of police support.

Close cooperation with all levels of law enforcement agencies is essential. You'll need to carefully explain the nature of the problem, what your losses have been, and how your problem relates to existing laws. Policies by law enforcement agencies aren't established for these offenses and the police likely will be hesitant to commit the resources to open a full investigation. This can sometimes be overcome by persistently explaining the need for support, and by doing as much of the work as possible yourself.

It's important to communicate with the law enforcement community early. Determining what agency to contact may be difficult; within any organization, probably only one or two people can appreciate the nature of the crime being committed. For this reason, you may find it fruitful to explain your problem to all possible agencies, and allow them to refer you to the correct organization. Each law enforcement organization has its own specialties; don't assume that you'll necessarily get more support from a national agency -- indeed, you may find that your local police are far more interested and supportive.

When telephone traces are needed, advance arrangements with your police contacts will prove invaluable: the telephone companies generally require court-orders, and the police know how to work with the appropriate district attorney to obtain these. The phone companies, in turn, are comfortable reporting to the police, but do not wish to report to injured parties, for fear of lawsuits.

Conclusion: Locking the Barn Door

You can trace connections back to the source in most circumstances. You'll need to keep detailed records, analyze audit trails, set monitors, traps, and alarms, and closely watch your operating system. Actual line traces may be in real-time or historical. Ironically, there are relatively few technical challenges; the main problems are in coordinating the efforts of many organizations.

Once you've tracked the rascal, finding out just who's been giving you such grief, you'll still have to close all the doors, whether or not he's been prosecuted (or even arrested). You'll need to simultaneously delete the accounts which have been compromised, eliminate the security holes, and change all passwords on your system⁺⁺⁺. Pulling the plug may be quite involved, and calls for advance preparation

Our networks provide rich connectivity; alas, but few people are paying attention to the risks which are created. When confronted with attacks and intrusions, system managers often talk about tracing the connections, but seldom actually initiate traces. With a little perseverance, it's possible to unwind connections, and find out who's at the other end. After tracing an intruder, tell your tale to others -- let the rest of the world learn from your sorrows.

⁺⁺⁺ With hundreds of users, a complete password change is distasteful. Alas, but no other method (password aging, account expiration, account requalification) can assure you of a clean system, without compromised accounts.

Acknowledgements:

I am grateful to the U.S. Dept. of Energy for supporting this work through contract DE-AC03-76SF00098. For suggestions, comments, and critical reviews, I am indebted to Marv Atchley, Bruce Bauer, Rick Carr, Dave Farnham, Mike Gibbons, Roy Kerth, Steve Kougores, Dave Jones, Martha Matthews, Sandy Merola, Bob Morris, Ken Sebrell, Phil Sibert, Dave Stevens, Steve White, Regina Wiggins, and Hellmuth Wolf. For the best enchilada in the West, I'm grateful to Garcia's Restaurant in Albuquerque.

References:

1 Relevant codes include 18 U.S.C.A. 1030 (The Federal Computer Crime Statute); 18 U.S.C.A. 1343 (Wirefraud Codes); 18 U.S.C.A. 2518 (Interception of Electronic Communications). For an example of a modern state statute, see CA Penal Code S. 502 (1986)

2 Anecdotal stories abound of intruders obtaining system privileges. See, for example, 2600 Magazine Vol 4, January 1987, page 6-7.

3 More complete auditing tools are being proposed, especially for Unix. *viz*: J. Picciotto, The Design of an Effective Auditing Subsystem, 1987 IEEE Symposium on Security and Privacy, April, 1987.

4 Secure networks are receiving increasing attention, and a lively debate has grown as to how to address these knotty problems. See D. Nessett, Adding Central Authentication to DECNET, DOE 10th Computer Security Group Conference, May 1987, page 56.

5 For a general review of Local Area Networks, see R. Stallings, Editor: IEEE Tutorial, Local Network Technology, 1985.

6 The need to prepare in advance for computer break-ins, and to create routine responses to them is noted by B. Reid, Reflections on Some Recent Widespread Computer Breakins, Comm. ACM, Feb. 1987.

Milnet protocols are amply documented in the DDN Protocol Handbook series, NIC-50004, 50005, and 50006, available from the DDN Network Information Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025.

TOWARDS THE ELIMINATION OF THE EFFECTS OF MALICIOUS LOGIC: FAULT TOLERANCE APPROACHES

MARK K. JOSEPH
Computer Science Department
University of California, Los Angeles

ABSTRACT—Malicious logic can be placed into a computer system's software in order to deliberately disrupt normal operation. Of particular concern is its potential effect on military systems. Two possible effects of malicious logic, which are addressed in this paper, are denial-of-service and compromising data integrity. Presented are several ad hoc, admittedly imperfect, techniques that are designed to reduce the risk posed by malicious logic. These techniques can be used now, while more complete solutions are sought.

1. INTRODUCTION

Recently, several authors have observed many similarities and interrelationships between fault tolerance and computer security [6,19]. In fact, in [6], denial-of-service is viewed as the classical unreliability problem. The main goal of this paper is to explore the application of several fault tolerance techniques to the elimination of the effects of malicious logic. Additionally, for a more complete discussion, a few techniques from computer security are also included.

As presented in [3], a few definitions are essential. A fault is a hypothesized cause of an incorrect state of some system resource. An error is the erroneous state of that resource. A failure occurs when the user of a computing system observes the system not performing as was specified.

The application of fault tolerance techniques to the problem of malicious logic is derived from the observation that its effects can be classified under the fault class of "by intent". This class of faults includes both accidental and deliberate faults. Here, malicious logic are deliberate design faults (in software or hardware) that cause errors in a computing system which may lead to improper service.

The techniques presented in this paper are only for highly critical systems. They address the threat of a trusted engineer inserting malicious logic into the computing system he or she is developing or maintaining.

Other recent work addressing protection techniques against malicious logic appears in [5,13,17]. These techniques could be used in conjunction with those presented here. Definitions, examples, and derivation of fundamental principles of denial-of-service in operating systems and computer networks appears in [7,8].

2. MULTI-PRONGED DEFENSE

Malicious logic can disrupt a computer system's normal operation from many locations in its software (e.g., application and operating system code). This large search space provides many opportunities and makes it difficult to prevent or detect malicious logic insertion into software. A multi-pronged defense, composed of off-line and on-line techniques, is proposed to reduce the risk posed by malicious logic. Off-line techniques (e.g., verification) are directed at preventing the insertion of malicious logic for the entire life-cycle of software. On-line techniques (e.g., execution monitoring) attempt to counter the effects of malicious logic that has successfully made its way into a deployed computer system.

All the on-line techniques presented are completely application dependent, whereas the off-line techniques can be more general. The reason for this lies in the fact that the on-line techniques are used in single threat counter-measure pairs, whereas each off-line technique can cover many threats.

Tradeoffs of performance versus the degree of risk are essential and should be carried out from the onset of a project. Additionally, to determine the effectiveness of a chosen collection of ad hoc techniques, an error seeding approach directed by penetration teams could be used. Each of these topics is examined in detail below.

3. PROPERTIES OF MALICIOUS LOGIC

Malicious logic is defined as deliberate design faults with the intent to commit an unlawful act or cause harm

without legal justification or excuse. Two possible effects of malicious logic, which are addressed in this paper, are denial-of-service and compromising data integrity.

Malicious logic is designed to avoid detection by both off-line and on-line techniques. To escape detection by off-line techniques, malicious logic is hidden in the complexity of the system's software or hardware. For example, in software this can be done by the use of multiple levels of macro calls, and deliberate use of complex and tricky coding practices. To hide from on-line techniques, malicious logic could try to create errors which appear to be results of naturally occurring faults.

4. ON-LINE TECHNIQUES

On-line techniques include additional software, hardware, and partitioning methods aimed at preventing the effects of existing malicious logic in a deployed computer system. It would be useful if, during execution, these techniques could also explicitly determine the location of such logic. Once located, it could then be targeted for removal as soon as possible.

A. N-Version Programming

N-Version Programming (NVP) can be used to provide reliable software [1]. N-versions of one program are independently designed and implemented from a common specification (or possibly from several independently written specifications). All N-versions are executed concurrently, typically on an N-processor computer system. During execution, the versions periodically vote on intermediate results and on the final result. As long as a majority of versions produce correct results, design faults in one or more version will be masked out. The strength of this approach is that reliable computing does not depend on the total absence of design faults.

Thus, NVP can be used to maintain the integrity of function and data by masking out the incorrect outputs of deliberate design faults. The probability of identical copies of malicious logic appearing in a majority of the N-versions is diminished due to the independent design and implementation of multiple versions.

A topic of further research is whether NVP is effective against denial-of-service threats. At this time, the following observations can be made. Instances of denial-of-service threats which involve the hoarding of system resources (e.g., CPU time, disk space) may be

prevented by NVP. The specification(s) of the N-versions must clearly state a set of restrictions that all versions must adhere to. For example, it can be specified that a version can have only a limited number of open files and/or child (forked as in UNIX) processes. (Each child consumes CPU time, main memory, and disk space.) Now, the voting mechanism used in NVP can be applied to a version's actions, such as system calls made, rather than to generated data values only. Thus, if less than a majority of versions try to obtain excess resources, the remaining versions will prevent such hoarding by masking out the resource requesting system calls.

A new instance of the denial-of-service threat may be possible for 2VP systems. Malicious logic need only be placed in one version, and would be designed to deliberately cause the two versions to disagree. Typically, a majority of versions is needed in order to produce a result. Thus, continued disagreement could cause some degree of denial-of-service.

At least two solutions exist for this new problem. The above example emphasizes an important feature of most NVP systems, that of masking. Only when N is greater than or equal to three can incorrect actions be masked out. Thus, one solution is to prohibit the use of 2VP systems.

Another solution is to use a hybrid form of NVP and Recovery Blocks [1,3] to prevent the malicious version from voting and forcing a disagreement. This is done by adding trusted self-checking code (i.e., the acceptance tests used in Recovery Blocks [15]) to both versions. Acceptance tests are additional program statements which are used to test whether a section of code performs as it was specified. Each time the malicious version failed an internal acceptance test its outputs would be ignored, thus preventing the denial-of-service. Such a hybrid form has already been shown to be effective for handling accidental design faults in NVP systems [3].

It is noteworthy that NVP also addresses completeness which is part of integrity, and timeliness of action. Several versions ensure through voting that all specified actions are performed. A timeout mechanism on a voting point prevents prolonged periods without action. Timeliness of access to some specified computing system service is an important capability in combating denial-of-service threats [8].

Additionally, the acceptance tests in the hybrid form of NVP and Recovery Blocks could attempt to distinguish deliberate and accidental design faults. Thus, detection of

deliberate design faults could be used to trigger an alarm notifying the appropriate authorities. It appears that more than just masking out design faults is needed if locating deliberate design faults is also desired. It should be made clear that in general all design faults are important. However, this discussion concentrates only on deliberate ones.

NVP is application dependent in two ways. First, determining how much, and which parts, of a software system will be built using NVP may be different for each application. Second, if used against denial-of-service threats, then restrictions placed in the specification(s) will likely be different for many applications.

B. Software Safety

Software safety techniques [10,11] have been applied to safety critical systems. An entire system view is taken in applying these techniques (i.e., both computer and non-computer hardware). System conditions which could lead to unsafe failures, called hazards, are hypothesized. Fault-tree analysis is used to locate where, if at all, in the system's software these series of conditions could occur.

Safety assertions are used to detect hazards and are a form of the acceptance test used in Recovery Blocks. Safety assertions are placed in the software along with recovery routines which are used to restore a system to a safe operating or fail-safe state. The strength of this method is in the total system view taken.

These techniques are also applicable to prevent the effects of malicious logic. A typical example of denial-of-service is an overloaded use of a system's processing resources (e.g., CPU time). Here the unsafe failure state is denial-of-service, while the hazard is the overloaded processing.

Assume (for this example) that fault-tree analysis determines that in the executive's scheduler this hazardous state could be observed. To counter this threat, the following safety assertion would be placed there:

```
assert underload: if utilization <= max_limit
on failure do
    assert diagnosis1: [condition]
    assert diagnosis2: [condition] od
```

When the "underload" assertion becomes false, special recovery routines will be invoked via the "on failure do" clause. These routines are application dependent and can be grouped in a safety executive as described in [11].

To handle this possibly intentional overload condition the recovery routines would preempt running tasks. This should continue until the load on the system's computing power decreases to a point where real work can progress.

C. Monitors

Let us consider a large banking institution's transaction processing system as a target of malicious logic. In the peak of business activity, the bank's computer network of automated teller machines and mainframes is forced into a self-test operational mode. These tests could require such a significant amount of computing power that the bank's computers are unable to process any significant number of incoming transactions.

This situation could result in a large financial loss to the bank in question. In fact, the bank could be held for ransom, such that its computers would occasionally be rendered inoperative unless a sum of money were paid. To counter this particular threat and, possibly, others like it, a trusted computing base (TCB) can be defined that mediates actions which are meaningful at the application level [4, p.67]. Access to objects involves not only reads and writes, but how and when application and operating system functions are invoked. Here, programs are the objects, and access to them is equated to their execution.

Now, invocation of the self-test function can be accomplished only after the TCB scrutinizes the request. All such potentially damaging use of basic system functions can be placed behind this defined security perimeter. All requests which are disallowed can then be viewed as auditable events. This technique requires defining a different security perimeter for each application. The potential for misuse of system functions is typically different for each application.

The concept of program flow monitors (PFM) [18] can be extended in order to prevent incorrect actions of a program on data items. To do this, each of the defined data manipulation functions (e.g., remove network packet header) is given a unique signature; for example, a sequence of bits in a bit vector. Also, each data item is initially given an empty signature. The result of a sequence of data manipulations is a combination of all the performed functions' signatures stored in the data item's signature.

For each sequence of acceptable data manipulations, an associated sequence of acceptable signatures exists and is stored in the PFM. That is, one signature exists for the

result of each data manipulation. At the application of a data manipulation function, the PFM precomputes what the resultant signature will be if the operation is performed. This precomputed, dynamically generated signature is tested by the PFM to see if it represents a valid signature. If the signature is not acceptable, then the data manipulation is not performed and some response depending on the particular application is necessary (e.g., auditable event, drop packet).

To strengthen this approach, the number of times that the same function is applied to a data item can also be encoded in the signatures [12]. An example of where this can be used is a network protocol function that removes a packet header. Correctly functioning protocol software should remove the header only once. However, malicious logic may try repeatedly to remove the header in order to obtain a packet's data. Assuming that the protocol software is not authorized for access to the packet's data, such access would generate an invalid signature.

Thus, program flow monitors can be used to ensure the integrity of function and data. This could also be viewed as just another example of part of a TCB, as mentioned above.

5. OFF-LINE TECHNIQUES

Off-line techniques are used to remove malicious logic throughout the entire software life-cycle (e.g., the development, testing and maintenance stages of a project). Removal of malicious logic follows its explicit detection in a software system.

These techniques are applied to all types of software in a computer system (e.g., both operating system and application code). In particular, the on-line security mechanisms chosen to protect a system from attacks are themselves targets for malicious logic insertion. The trust placed in these mechanisms must be validated. This can be done by one or some combination of the following methods: fault-tree analysis as mentioned above, formal verification, testing and code reviews.

A. The Use of Formal Verification

Current formal verification techniques and tools can effectively examine only small pieces of software (e.g., a security kernel in an A1 certified computer system [4]). If the security mechanisms used are too large, then formal verification can be done on selected pieces.

For NVP, formal verification or any validation method should be concentrated on the support software, since this is where malicious logic could have its effects. For example, parts of the DEDIX [1,2] (DEsign Diversity eXperiment) system developed at the UCLA Center for Experimental Computer Science should be formally verified (e.g., the voter logic). If each version of the support software was itself from a diverse design, then the importance of verification could be reduced.

For software safety techniques, the safety assertions and recovery routines are candidates for formal verification. Finally, the same extensive methods used for TCBs seem to apply to all types of monitors.

B. Testing

Malicious logic could be designed to trigger on particular state conditions (e.g., the date, or by command). The trigger could also be disabled until a command was sent enabling it. This enabling command could simply be a sequence of legal but odd system requests (e.g., one hundred health status system requests in a five-minute time interval).

Standard testing methods would likely be ineffective in locating such malicious logic, since they would probably miss enabling the triggers. Therefore, new testing approaches aimed at detecting possible enabling command sequences (i.e., channels) and trigger devices should be used. This requires a separate test plan from the normal functional testing.

In addition, the use of independent testing teams from alternate contractors has been shown to increase testing effectiveness. Component testing, at the module level by the independent teams, also seems necessary, since testing at only the device level is of insufficient depth for our purposes.

C. Configuration Control

Very strict configuration control software and procedures are **essential**. This will help to ensure that malicious logic is not added after all tests are made to ensure its absence. To guarantee that proper procedures are followed, surprise inspections could be used in order to monitor the developer.

D. Code Reviews for Malicious Logic

It is a straightforward extension to perform code reviews

specifically to discover malicious logic. This review process should be done by teams in order to ensure its validity. Reviews are conducted during the development process rather than afterwards by penetration teams.

These last two techniques (i.e., configuration control and code reviews) can go a long way to prevent insertion of malicious logic into a software system. It is this author's opinion that they should always be a part of the selected off-line techniques.

6. TRADEOFFS

It is frequently difficult to satisfy all the desired objectives of a system (e.g., performance, security, fault tolerance, compactness, etc.). Since resources are always limited, it is essential to decide from the onset of a project the amount to dedicate to security concerns. Resources are both computer resources, such as millions of instructions per second, and project resources, such as a budget to perform verification.

To determine the amount of resources to dedicate, an acceptable degree of risk from the threats posed by malicious logic must be defined. Tradeoffs should be performed between security and other desired system objectives using the acceptable degree of risk as a control on the investment in security mechanisms. Of course, in order to perform these tradeoffs, some idea of the costs and effectiveness of the proposed security mechanisms must exist.

Additionally, an analysis of the proportion of off-line versus on-line techniques to be used in the total security budget should be performed. Decisions of this type can be made based on the cost, effectiveness, and performance impact of each approach. For example, NVP can be very expensive in development and maintenance. Therefore, widespread use of this technique in certain software systems may be unlikely. Instead, it could be used selectively, as determined from a tradeoff study.

Obviously, off-line techniques have the advantages of not affecting performance, weight, or power (i.e., attributes of the physical computer system). However, on large software programs, their effectiveness may be too limited. This is evident from experiences with current formal verification technology.

7. MEASURE OF EFFECTIVENESS

Effectiveness measurements of the collection of ad hoc approaches chosen can be used in design refinement. If serious protection problems are discovered during measurement, steps can be made to compensate.

To obtain this measure, deliberate insertion of malicious logic similar to the technique of fault injection used in fault tolerance to determine fault coverage [18], and error seeding techniques used in software testing [14], can be employed. The determinations of what malicious logic and where to implant it can be made in several ways. First, use a penetration team whose experience in security concerns helps them devise implants. Second, test specific conditions addressed by existing on-line security mechanisms. And last, use analysis techniques such as fault-tree analysis as employed in software safety [10,11].

Deliberate insertion of malicious logic for testing purposes obviously must be done with great care. The process of implanting must be well documented and performed by a team. Implants should be placed only in experimental versions used solely for testing. These versions should never be placed in the same configuration library where the real operational software is stored.

Measurements of effectiveness of both on-line and off-line techniques can be performed. For on-line techniques, malicious logic is placed in operating software during normal system testing. For off-line techniques, such as formal verification, malicious logic is deliberately placed in preverified code during early design stages.

Performing such measurements requires defining what is to be measured (the metric) and how the results are to be evaluated (the criterion). The metric is the percentage of instances, where implanted malicious logic goes undetected out of the entire body of tests [18]. The criterion is composed of two parts. First, it must take into account the coverage, or quality of the error seeding cases [14]. Second, the calculated percentage is interpreted relative to the acceptable degree of risk defined in the design phase. Two results should be generated: one for all on-line, and one for all off-line techniques used. This way, the return on investment of each approach can be compared.

8. CONCLUSIONS

The multi-pronged defense presented is meant to be a practical and immediately usable approach to decrease the

risk of malicious logic in critical computer systems. All the techniques presented appear to be used in specific threat-countermeasure pairs. It may be possible to strengthen these techniques by changing this one-to-one relationship to a many-to-one relationship (i.e., one countermeasure covering many forms of malicious logic).

In approaching the difficult problem of preventing denial-of-service and ensuring integrity, new ad hoc techniques should be devised. These new techniques should be encouraged but still must follow good security common sense. For example, new techniques should not depend on trusting large portions of software. It is beyond the current state-of-the-art to formally verify and validate trust in large pieces of software. Also, such obvious weaknesses will turn into the target of the malicious logic it was meant to prevent.

It is important to recognize that several of the ideas presented in [7] support a fault tolerance approach to the denial-of-service threat. These ideas include the detection of, and recovery from, denial-of-service. Recovery may require the use of redundant services in order to maintain proper service. These ideas are straight-out of accepted fault tolerance concepts.

Current research is directed towards extending the schemes presented, determining their effectiveness, devising additional extensions of fault tolerance techniques, and analyzing the similarities between fault tolerance and computer security. One example of an application domain which must be addressed are Database systems. These present many additional opportunities (e.g., via locking) to cause denial-of-service [9].

The connection between fault tolerance and computer security has been made by the observation, that the effects of malicious logic can be classified under the fault class of "by intent". The use of design fault tolerance and NVP is alluded to in [6], however, the possibility of the deliberate nature of the faults is not mentioned. This paper has taken a first look at how design diversity could be used against malicious logic.

APPENDIX

Background Devious Actions

Can a Trojan Horse, inadvertently used by an NVP system, perform devious actions in the background while producing valid results to be voted on? This question certainly needs much more attention, but initially the following points are made.

1) The whole idea of NVP is that many of a version's actions (e.g., calls made as well as data generated) are voted on. Thus, these background devious actions will either be masked out entirely or severely limited. An obvious trade-off between degree of risk and performance exists here.

2) Input to each version of an NVP system needs to be obtained from different sources. If each version obtains the same bad data, then the masking capability of NVP could be defeated. By analogy, if each version of an NVP system calls one version of a common program that contains a Trojan Horse, then masking out its devious actions will not be possible.

3) A multi-pronged defense is advocated. Thus, a collection of on-line and off-line techniques should be used. If one technique fails to detect and prevent a devious action then it is hoped that others will catch it. This concept is very similar to the idea of hierarchical fault recovery in fault tolerance [16].

ACKNOWLEDGMENTS

Special thanks to S.Glaseman and G.Gilley of The Aerospace Corporation for their many helpful comments on this paper.

REFERENCES

- [1] A.Avizienis, "The N-Version Approach To Fault-Tolerant Software," IEEE Trans. on Soft. Eng., Vol. SE-11, No. 12, Dec. 1985, pp. 1491-1501.
- [2] A.Avizienis et al., "The UCLA DEDIX System: A Distributed Testbed For Multiple - Version Software," 15th Annual Int'l. Symp. on Fault-Tolerant Computing Systems, Ann Arbor, MI, June 1985, pp.126-134.
- [3] A.Avizienis, and J.P.J.Kelly, "Fault Tolerance By Design Diversity: Concepts And Experiments," Computer, Vol. 17, No. 8, August 1984, pp.67-80.
- [4] Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28 - STD, Dec. 1985.
- [5] D.E.Denning, "An Intrusion - Detection Model," 1986 IEEE Symp. on Security and Privacy, April 1986, pp.118-131.
- [6] J.E.Dobson, and B.Randell, "Building Reliable Secure Computing Systems From Unreliable Insecure Components," 1986 IEEE Symp. on Security and Privacy, April 1986, pp.187-193.
- [7] V.D.Gligor, "Denial-of-Service Implications for Computer Networks," Proc. DoD Computer Security Center Invitational Workshop on Network Security, March 1985, pp.9-33 - 9-48.
- [8] V.D.Gligor, "A Note On The Denial-of-Service Problem," 1983 IEEE Symp. on Security and Privacy, April 1983, pp.139-149.
- [9] R.R.Henning, and S.A.Walker, "Computer Architecture And DataBase Security", 9th National Computer Security Conf., National Computer Security Center, Sept. 1986, pp. 216-230.

[10] N.Leveson, "Software Safety: Why, What, and How," ACM Computing Surveys, Vol. 18, No. 2, June 1986, pp.125-163.

[11] N.Leveson, "Software Safety",- Chapter 7, Resilient Computing Systems Volume 1, Editor: T.Anderson, New York: John Wiley & Sons, 1985.

[12] S.Osder, "The DC-9-80 Digital Flight Guidance System's Monitoring Techniques," AIAA Guidance and Control Conf., August 1979, pp.64-79.

[13] M.M.Pozzo, and T.E.Gray, "A Model For The Containment Of Computer Viruses," AIAA / ASIS / DODCI 2nd Aerospace Computer Security Conf., Dec. 1986, pp.11-18.

[14] C.V.Ramamoorthy, and F.B.Bastani, "Software Reliability: Status and Perspectives, IEEE TSE, July 1981, pp.354-371.

[15] B.Randell, "System Structure For Fault Tolerance," IEEE Trans. on Soft. Eng., Vol. SE-1, No. 2, March 1975, pp.220-232.

[16] D.A.Rennels, "Fault-Tolerant Computing — Concepts And Examples," IEEE Trans. on Comput., Vol. C-33, No. 12, Dec. 1984, pp.1116-1129.

[17] R.R.Schell, and D.E.Denning, "Integrity In Trusted Database Systems," 9th National Computer Security Conf., National Computer Security Center, Sept. 1986, pp.30-36.

[18] M.A.Schuette, and J.P.Shen, "Processor Control Flow Monitoring Using Signed Instruction Streams," IEEE Trans. on Comput., Vol. C-36, No. 3, March 1987, pp.264-276.

[19] R.Turn, and J.Habibi, "On the Interactions of Security and Fault Tolerance," 9th National Computer Security Conf., National Computer Security Center, Sept. 1986, pp.138-142.

THE SETUID FEATURE IN UNIX® AND SECURITY

Steve Bunch

Gould Computer Systems Division
1101 E. University
Urbana, Ill. 61801
217-384-8515

srb@gswd-vms.arpa
ihnp4!uiucdcs!ccvaxa!srb

1 INTRODUCTION

The UNIX® system^{1,3,11} contains a simple, elegant, and powerful feature called SETUID [U.S. Pat.# 4,135,240]. This feature allows a user to temporarily assume the identity of another user and obtain the discretionary access rights, and the privileges, of that user. This feature is used to control privileged operations and to build protected subsystems. It is invoked by giving a program the SETUID property. Upon execution of such a program, any user executing the program acquires the access rights and privileges of the owner of the program. It is the responsibility of the setuid program to prevent abuse of the additional access rights it grants. This paper informally describes some of the security implications of this facility, and describes several alternatives which can provide similar functionality with better security.

The first section of the paper defines some important terms. (We assume basic familiarity with UNIX, but not with the SETUID/SETGID concepts.) Next, the paper examines some of the properties and uses of this mechanism, examines some of the security implications of it, and finally discusses alternative methods of providing similar or equivalent functionality.

2 DEFINITIONS

In this section, we define some of the important terms and concepts used in the remainder of the paper. There are several slightly different implementations of the system calls and semantics of SETUID/SETGID in different flavors of the UNIX system; we have tried to keep the points made in this paper generic and applicable to virtually all of them. We will ignore some details and complications which are of limited interest in order to keep the discussion simpler.

In UNIX systems, user names are mapped onto integers known as *user IDs* during the login process. This mapping can be assumed to be one-to-one for this discussion. One distinguished user, the *root* or *superuser*, possesses all privileges to perform sensitive operations in UNIX.

Largely as a convenience in managing access to files, and in some systems for accounting purposes, users belong to one or more *groups*. A group is simply an arbitrary list of users who are treated together for access control purposes. A user is associated with a *default group* upon login, and can change that association during his login session. (There are several variations of this in different UNIX systems, including simultaneous membership in

multiple groups. We will follow the original UNIX convention of single group membership.)

Processes in the UNIX system follow the intuitive definition of a process. A process possesses many properties, but there are a few which are especially important to this paper:

- *Real User ID* (RUID). This is the user who is the actual owner of the process, i.e., the user from whose login process the process is descended.
- *Effective User-ID* (EUID). This is the user whose discretionary access privileges are currently available to the process. It is normally the same as the RUID.
- *Real Group-ID* (RGID). This is originally the default group associated with the real user id. It can be changed explicitly by the user to any of the groups to which the user is authorized (this is done differently in different versions of UNIX - the distinction is unimportant here).
- *Effective Group-ID* (EGID). This is a group whose discretionary access privileges are currently available to the process. It is normally the same as the RGID.

We will use the notation

RUID(process)
EUID(process)
RGID(process)
EGID(process)

to indicate the UIDs or GIDs of a particular process.

The discretionary access control (DAC) mechanism of UNIX is implemented by associating three sets of *mode bits* with an object being controlled. (When discussing objects in this paper, we will generally refer only to files; the conclusions generally apply to other objects protected by owner/group/other mode bits.) All such objects have an *owner* and *owning group*, which generally describe the user id and group the owner belonged to when the object was created. Each set of mode bits consists of three yes/no permission bits which if set allow *read*, *write*, and *execute* access (other permissions, such as *directory search*, are overloaded onto the same three bits). The three sets of mode bits describe the access permitted to *owner* (the file owner), *group* (users in the same group as the file owning group), and *other* (all other users and groups). If the owner is attempting access, only the *owner* bits are checked. If someone in the owning group other than the

owner is attempting access, only the *group* bits are checked. If the attempt is by neither, the *other* bits are checked. The EUID of a process attempting to access an object is used in determining if a requested access is being made by the owner of the file; the EGID is used in determining if the attempt is being made by someone in the owning group of the file.

The SETUID mechanism is invoked by setting the SETUID property on a file. This property is externally applied, like mode bits. Upon executing a program file with the SETUID bit set, the EUID of the resulting process is set to the owner of the program file. The SETGID mechanism works similarly, but the EGID of the process is affected. SETUID and SETGID can be used separately or together on an executable file.

It is useful to be able to talk succinctly about the set of files to which a user, group, or process has access. We describe these sets with the following notation:

FILES(*mode, user*)
FILES(*mode, group*)
FILES(*mode, process*)

This construct represents the enumeration of the entire set of files in the system which can be directly accessed in the way described by *mode* (e.g., read, write, execute, search) by that *user, group, or process*. The definition of "accessible" must be made very carefully, as we describe in the later "Security Implications of SETUID" section. We will use the operator "+" to represent the set union operation when discussing operations on these sets.

3 PURPOSE OF THE SETUID/SETGID MECHANISM

The SETUID/SETGID mechanism is used in practice for two functions, which are quite distinct and separable but are often confused:

1. SETUID/SETGID is used as the privilege-granting mechanism in UNIX.

There are many "privileged" operations in UNIX. Control of these is coarse-grained: one specific user, the "superuser", has all privileges. In addition, control over certain very special system files (e.g., /dev/kmem, the pseudo-file representing the memory image of the kernel) is often invested in SETGID programs, which may perform sensitive operations on those files. (Some of these operations are privileged because of the damage they can do if misused, not because of the inherent sensitivity of the operation. For example, for a user to simply list his own active processes requires accessing the kernel memory image in a standard UNIX system.)

2. SETUID and SETGID are used to build protected subsystems.

A protected subsystem in UNIX is implemented as a SETUID/SETGID program (or family of programs) which control access to a file or files owned by the same user/group as the program(s). Any user executing such a program temporarily acquires access to the files, but all his accesses are mediated by the program. Examples of such subsystems include mail systems (which protect the mail data base), data base systems, bulletin board systems such as notes and news, games (which protect scoring information), other software packages which want to keep statistics or other side

information which they do not want users to be able to access except from inside the package, and programs which wish to implement their own discretionary access policy.

The operating system kernel itself provides no finer control over its privileged operations than the superuser privilege, which is all-powerful. All processes with an EUID or RUID equal to that of the superuser (for example, a process generated by executing a SETUID program which is owned by the superuser) possess all possible kernel privileges and can use them or abuse them as they wish. It is the program itself which must correctly do only those privileged operations that are consistent with the security of the system. For example, it is the responsibility of the *login* program, a SETUID program owned by the superuser, to verify that a user presents the correct password for a specific UID before it grants the user access to the system as that user.

Some privileged operations are implemented in user-level processes by SETUID/SETGID programs. These programs are sometimes superuser programs, but need not always be. For example, some set the EUID to a special non-superuser UID or set EGID to a special GID, where the UID/GID is the owner of some special privileged object. Used in conjunction with the UNIX discretionary access mechanism, this provides a quite distinct mechanism from the kernel-arbitrated privileged operations mentioned above. This type of privilege typically involves the manipulation of the special file system objects like the /dev/kmem or /dev/mem pseudo-files, which permit access to all internal data structures of the operating system kernel. (For example, the UNIX *ps* command, which obtains the current state of all processes in the system regardless of owner, obtains its information in this way.)

SETUID/SETGID is a very powerful but coarse-grained privilege mechanism. In building a secure version of UNIX, it is necessary to minimize the potential for abuse of privilege. The potential for abuse of SETUID/SETGID mainly comes from the first point above, the acquisition of privileges with SETUID/SETGID. (A companion paper⁴ discusses a privilege mechanism to help control the granularity problem.) Abuse or incorrect usage of the mechanism to implement trusted or protected subsystems is also possible, as we discuss below.

4 SECURITY IMPLICATIONS OF SETUID

4.1 Interaction with Directory Search Rules

The general accessibility of files to a process in UNIX can be described as the union of files accessible to its user and to its group. Since there are potentially two different user id's and two different group id's, however, the details of this become complicated. Let us assume for a moment that the meaning of FILES(*mode,process*) is defined as the set of files which the *open* system call can successfully open with the specified mode. The basic rule for some specific mode *mode* in the set [*read, write, execute*] would appear to be

$$\begin{aligned} \text{FILES}(\textit{mode},\textit{process}) = & \\ \text{FILES}(\textit{mode},\text{EUID}(\textit{process})) + & \\ \text{FILES}(\textit{mode},\text{EGID}(\textit{process})) & \end{aligned}$$

(We assume that files available via "other" access permission are included in one or more of the component sets). There are two non-intuitive points that we would like to make about the above description.

1. The access rules for UNIX `open` specify that the access checks are done using *effective* user and group id's. However, by using an unprivileged system call (`setuid/setgid`) to set its EUID/EGID to its RUID/RGID, a process *can* in fact often access both sets of files. The extent to which this is possible is highly dependent on the mode settings on directories and on the nuances of the particular UID/GID manipulation primitives provided on the particular UNIX version being used. (It may in some cases require spawning sub-processes and other trickery.) Consequently, the *real* user and group entries must be included in this computation to cover this case. To take this into account, the corrected formula should include the terms

+ FILES(mode,RUID(process))
+ FILES(mode,RGID(process))

2. Under our working assumption that we define the semantics of FILES using UNIX's `open` system call, even the corrected definition above is incomplete. In fact, FILES(mode,process) can be a superset of the union shown above due to the interaction of directory search access rules, access modes, and the RUID/EUID and RGID/EGID. Because of the directory search rules in UNIX, it is possible to have otherwise accessible files that cannot be found because they are located in an unsearchable subtree of the file system. A different user may be able to find them, but be unable to access them. The combination of accesses provided by different effective and real id's can be used to locate, then access, such files. An example is shown below.

A curious feature of the use of mode bits to deny access can show up in computing the FILES relation. Files belonging to a user may be inaccessible to the user because of the way mode bits are checked. This intentionally permits an owner to explicitly deny himself or his group access to an object. If the *other* or *group* permissions allow access to the EUID/EGID of an object, then a SETUID/SETGID process could access files belonging to the user himself which he could not access otherwise.

In order to illustrate the second point above, we can create an example in which a SETUID process (running with non-equal EUID and RUID) can access files which a process owned and operated by either UID alone could not access (Figure 1). First, we create a directory `dir` which allows *search* access *only* to EUID. Within `dir`, we next place a directory `subdir`, which allows *search* access to both the EUID and RUID. Within directory `dir/subdir`, we place subdirectories and files (`file1` and `dir2` in Figure 1) to which only the RUID has access. The SETUID process first changes to directory `dir/subdir` as its working directory. Then, it uses the `setuid` system call to set its EUID to the old RUID. The files and directories in `dir/subdir` are now available to the process. A process belonging to the original EUID could not have accessed the files, though it could find them. A process belonging to RUID could not have found them.

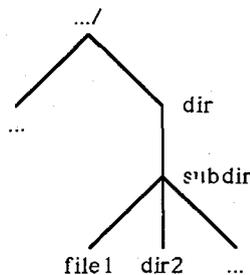


Fig. 1 - Gateway Directory Example

We refer to `dir` as a "gateway directory" later in this paper. (There are contexts in which this could actually be useful. For example, a mail directory could contain mail files owned by individual users, but be inaccessible except through a SETUID/SETGID program and a gateway directory owned by its owner/group. This minimizes the amount of privileged work that must be done by the SETUID/SETGID program, as it can simply change directories to the mail directory hidden beneath the gateway directory, reset its EUID/EGID to the RUID/RGID, and then proceed with its work using only the access permissions of the executor.)

These non-intuitive aspects of SETUID/SETGID appear to be potential security problems in practice only if administrators or users incorrectly try to isolate a subtree of the file system by a change in discretionary access permissions at its root. They also cause some problems with formal models, as discussed later.

4.2 Unexpectedly Granting Access to Files

In a SETUID/SETGID protected subsystem which obtains parameters such as file or object names from a user, the application must be extremely careful to insure that the operations performed by the subsystem correctly avoid violating the protection which the application is supposed to provide. This points out the general problem of *unintended grant of access* to the files of the owner (or owning group) of the SETUID (SETGID) program. Any files belonging to the owning user (owning group) must be protected by the application *in addition* to the file(s) that it is designed to protect. This means that a lot of care must be taken in the design of SETUID/SETGID programs; inexperienced programmers in particular should beware of using the SETUID property on programs which they allow others to use.

A classic example of this problem is the UNIX *mail* program of early Version 6 UNIX. The program ran SETUID to *superuser* in order to be able to write mail into a user's privately-owned mailbox. When an option was added to the mail program to allow a user to make it take its input from a file, a corresponding check to see if the user was supposed to be able to access that file was *not* added. Since the superuser can read any file, a user could obtain a copy of any file in the system by mailing it to himself. To make it easier to avoid this problem when writing a privileged program, UNIX System V now incorporates the notion of a *saved uid*, which allows a superuser process to set its EUID to its (probably non-superuser) RUID and later set it back. Using this capability, a program can temporarily suspend its privileges while performing commands requested by a user (and thus allow the kernel to perform its normal checks), and restore them later.

4.3 Trojan Horses

It is clearly possible for any program provided by another user to contain a Trojan horse. The program can access the files of the user who executes it, and can siphon off data and save it for the provider of the Trojan horse to peruse later. This is a generic problem with trusting *any* program written by another user, and is not specific to SETUID/SETGID programs. SETUID/SETGID does open the possibility of a new kind of Trojan Horse, however, as described below.

One method for a user to attempt to encapsulate a SETUID/SETGID process is to insure that the user's files are not accessible to the EUID/EGID under which the process operates, and therefore cannot exceed the bounds of the user's intent. However, one SETUID/SETGID program can execute another using the *fork* and *exec* system calls. Thus, unless the access of one SETUID/SETGID program to all other such programs can be totally denied, the user must protect against *all possible* EUIDs/EGIDs which the program might be able to assume. It is straightforward to plug this hole. For example, one can forbid an unprivileged SETUID/SETGID process to have a SETUID/SETGID process with a different EUID/EGID/RUID/RGID as a descendent (in any generation). (It is necessary to restrict this to unprivileged processes to permit programs like *login* to work as they do now.) Another approach was taken by IBM's Secure XENIX[®], which solves this problem by allowing SETUID processes to be executed only by a direct user command. (In that system, SETUID functionality still implements the privilege mechanism.) Either approach makes it impossible to accumulate access.

Few UNIX machines in environments with skillful system programmers and without routine housecleaning are free of hidden SETUID programs which make their owner the *superuser* or some other user¹⁰. (The names of such programs often start with a '.' so that they don't normally show up on directory listings, or use the same name as a valid SETUID program in the hopes of tricking a casual observer into thinking that they are simply copies of that program.) A useful feature added to Berkeley's version 4.3 UNIX system is the ability to disable SETUID/SETGID for specific mounted file systems. This makes it easier to insure that only authorized programs can perform this function, and makes importing foreign file systems much less dangerous.

Trojan horse code which a malicious user can arrange to have executed by a user whom he is trying to subvert can normally execute any SETUID/SETGID program available to that user, with arbitrary (presumably malicious) parameters. (This is one reason that a *trusted path* must be provided to all TCB services in a B3/A1 system.) The IBM Secure XENIX approach of allowing SETUID commands to be executed only by direct user command prevents this.

4.4 Modeling Issues

There are some security modeling issues which arise because of SETUID/SETGID. These issues may or may not present problems in practice, but they cannot be ignored.

- A process with the EUID of a user who isn't even logged in can get access to files. This is somewhat peculiar, as the effective user will not have passed the authentication procedures, and may not even be a valid user at the time of access. It is possible that this could result in a violation of certain security policies, e.g., policies which limit the times of day during which specific users are allowed to use the system. (Such a

limitation might be due to sensitive data being present on the system only during specific periods; the intent of such a policy could clearly be violated by a SETUID program.) It is non-trivial to validate the user represented by the new EUID at the time of execution of the SETUID program, as that would require the kernel to perform the revalidation and that function is now performed by user-level processes in UNIX. (One solution to this problem is discussed in section 5.2.)

- *Accurately* modeling discretionary access in an unmodified UNIX is inextricably related to the file system contents and the properties of all SETUID/SETGID programs in the system. Discretionary security models of UNIX can be simplified if they model the "maximum case" of access, classifying as accessible those objects whose access modes *would allow* access, should the object ever be reachable. (This leads to an overstatement of access in the security model.) However, this simplification does make modeling unusual encapsulation methods based on DAC (like the earlier example of "gateway directories") difficult or impossible.

There is no obvious simple way to accurately model the effects of all the SETUID/SETGID programs in a system on the discretionary security policy unless all possible inputs and effects of those programs are characterized. It will be interesting to see how implementors of B3 and A1 systems with SETUID/SETGID model it.

4.5 Integrity of SETUID/SETGID Programs

In order to prevent the subversion of a SETUID/SETGID program, UNIX systems must clear the SETUID/SETGID bits on program files any time a change is made to (at least) the file contents, owner, group, or permissions. This would appear to be a "given" in any UNIX system, but errors allowing SETUID/SETGID programs to be subverted have existed in many implementations in the past.

A popular way of influencing any trusted program is to provide it a false environment. Any trusted program which obtains information about its operating environment *from the user executing it*, either directly (e.g., names and parameters entered by the user), or indirectly (e.g., through the UNIX *environment* variables), must be careful what it believes. Subversion of programs by influencing their environment is a source of a large number of errors which have been found in SETUID/SETGID programs, especially large and complicated programs. One example of such a subversion occurs in programs which blindly execute a program name passed to them in an environment variable. For example, by changing the default name of a normally-trusted sub-command before executing such a program, a user can get a trusted program to supply data it should not, or even to execute a Trojan horse. Some of the subversions which are possible with existing programs in wide use are best left unpublished (especially those involving *SETUID/SETGID shell scripts*; some of these bugs are hard to fix). The only real cure for this problem is for trusted programs to operate in an environment which the user cannot influence, or for such programs to disbelieve their environment completely (or to verify it, where possible).

4.6 Interaction with Extensions to UNIX

It is non-trivial to foresee all the ramifications of SETUID/SETGID when extending UNIX. For example, Berkeley UNIX introduced the concept of allowing a process to operate with *multiple* groups simultaneously active. That is, a process in Berkeley UNIX has associated with it a list of GIDs, and access to an object is allowed if *any* of those groups is permitted that access. This is very convenient operationally, and facilitates sharing among groups. In this model, it is not obvious which group to associate with a newly-created file or directory. The approach taken by Berkeley is to use the owning group of the directory in which the object was created, *even if the process (and its user) are not members of that group*. The rules for setting the SETGID bit on a file allow the owner of a file to set it. If the owner sets the SETGID bit on a newly-created file to whose owning group he does not belong, the resulting SETGID program would allow the user access to files using the access rights of a group to which he does not belong. This bug was present in earlier releases (e.g., 4.1) of Berkeley UNIX, but the system now prevents this by insuring that the owning group on a file is contained in the set of groups of the process before the SETGID bit is allowed to be set. This cautionary tale indicates that the security ramifications of any change to the discretionary access mechanisms of UNIX should be examined carefully.

5 REPLACEMENTS FOR SETUID/SETGID

The remainder of this paper discusses three different methods to provide the functions performed by SETUID/SETGID on UNIX. These methods display decreasing degrees of modification to UNIX and display various degrees of compatibility with existing programs which use SETUID/SETGID (including essentially full compatibility). The third method presented is a part of a general-purpose privilege mechanism which we believe is superior in controllability and security to SETUID/SETGID while remaining compatible for most SETUID/SETGID programs. All three offer more secure control over the privileges in UNIX than unmodified SETUID/SETGID mechanisms, and all three allow the creation of protected subsystems.

5.1 Restricted Environments

Gould used a different approach from SETUID/SETGID for kernel integrity and control of privilege in its C2-rated UTX/32S UNIX system. When Gould began work on its system in 1985, there was considerable debate in the security community concerning the inherent security of systems which use the SETUID/SETGID concept. This debate is still not totally resolved today. While quite interested in its outcome, we were unwilling to wait until the debate was finished to implement our C2 UNIX system. So, in order to facilitate the formal evaluation of our system, we removed the SETUID/SETGID facility completely. This was a controversial decision. The lost functionality was implemented using trusted server processes, originally introduced as part of the integrity mechanism of the system, operating outside *restricted environments*.

The *restricted environment* is a concept based on the UNIX **chroot** system call, which restricts file system access of a process to a subtree of the UNIX file system. In Figure 2, we show a subset of a UNIX file system. The directory `/unpriv_re` forms the root of the unprivileged environment which is encircled in the figure. Users operating inside the environment have no way to locate files outside it. From the standpoint of a process operating inside the restricted environment, `/unpriv_re` is the root of its file system, and the directory `/unpriv_re/mnt` is addressed by the path name `/mnt`. Processes with this limited visibility of the entire system's file tree have no direct access to system-critical

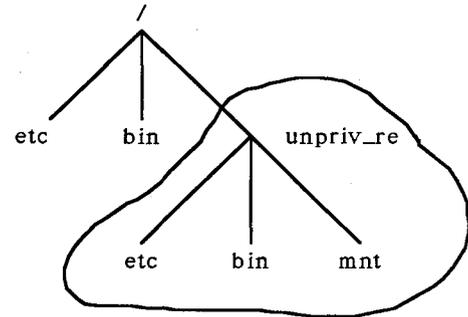


Fig. 2 - A UNIX File System with a Restricted Environment

files (e.g., the password file) or to privileged programs and their directories, all of which are outside the restricted environments. This makes it much harder for a user process in a restricted environment to affect the Trusted Computing Base (TCB) of the system in any way.

Protected subsystems can be implemented in at least two ways in this model.

1. Make the subsystem a server process, operating outside the restricted environment (Figure 3).

The server process need not be privileged in general, though the details of implementation in our system interfere with this (as an unprivileged process, it cannot use the trusted interprocess communication mechanism).

In Figure 3, we show a client program inside the restricted environment communicating with the subsystem, which operates outside it. They communicate via an interprocess communication mechanism known as *secure sockets*.

2. A second method is to run a trusted subsystem in a restricted environment of its own, with no users, along with all the files to which it is to control access (Figure 4).

Clients communicate with the subsystem via a server which arbitrates references and performs communications between server and client. We did not support multiple restricted environments on the first system release, so we have no field experience with this approach. Multiple environments will be a feature in future releases, so this method will be available.

This approach is capable of implementing *mutually suspicious subsystems*, and requires minimal or no change to application programs. It is a very strong method for isolating subsystems, and is probably the method of choice for operating large existing SETUID/SETGID applications such as databases where the extra isolation does not interfere with functionality. Figure 4 shows this case.

The privilege control mechanism we used with *restricted environments* is a simple one: processes must be born privileged, i.e., they must be designated by a privileged parent as inheriting the privileges of the parent when created. (The one initial

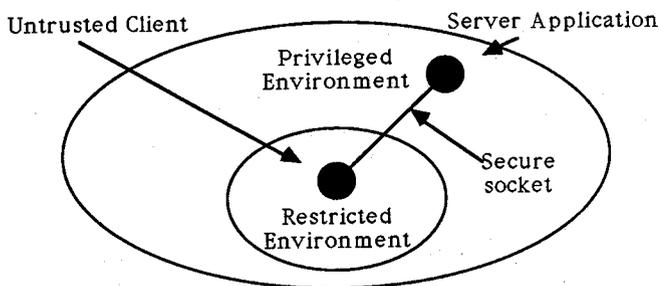


Fig. 3 - An Application Implemented as a Server

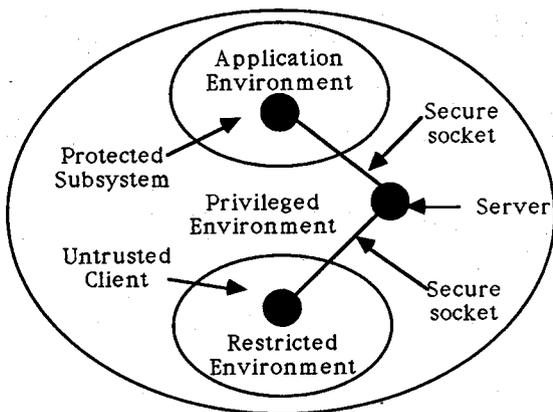


Fig. 4 - A Server Mediating between Client and Application

process is privileged.) All privileged operations are executed by privileged server processes which are created to serve that function and that function alone. Privileged servers operate outside the restricted environment subset of the file system, and therefore system files are visible to them. Because the original UTX/32S is a C2 system, it was not deemed necessary to subdivide the system privileges (i.e., implement a least-privilege TCB). So most privileged processes still run with essentially superuser privileges in this model.

When using server processes to perform functions formerly performed directly by a privileged system call, a set of subroutines can be provided to make some of the communications invisible to clients. The subroutines establish communication with appropriate privileged processes, pass parameters to it, and generally behave as if they perform the service directly. If such subroutines are substituted for a few key system calls and service subroutines (e.g., the password validation function), many programs need only be recompiled in order to run.

One of the first questions people ask about the use of server processes is its performance. It turns out that a few operations actually go faster, since the server process already exists and a connection to it is very cheap to make. Other operations must still create an additional process, so they are somewhat slower. Overall, there is usually little difference in performance between

this system and the non-secure UTX system used as its base. (A more important performance drain comes from auditing - a function that all secure systems have to implement.)

5.2 Restricted SETUID/SETGID

An extension to the concept of restricted environments is to allow the limited use of SETUID/SETGID within a restricted environment. This SETUID/SETGID with modified semantics would not permit a process to acquire any system privileges, but would allow the controlled access to data files as SETUID/SETGID does now. In other words, it would be used only for implementing the protected subsystem component of the SETUID/SETGID functionality described in the early sections. This would allow the vast majority of SETUID/SETGID application programs to work. (Those which also perform privileged operations reserved for the superuser would not be able to execute the privileged system calls required, and would need to be modified).

One of the weaknesses of SETUID/SETGID, even in this limited form, is the transitive acquisition of discretionary access by executing a chain of SETUID/SETGID processes, as described in section 4.3. It can be limited as described there.

As described earlier, another weakness of SETUID/SETGID is that such programs can provide accidental access to files owned by the EUID/EGID other than those it is supposed to. Three ways of avoiding this problem come to mind:

1. Careful programming. (This can also be described as "wishful thinking".)

Programmers must keep a long list of cautions in mind when building SETUID/SETGID programs¹⁰. Knowing as many case histories as possible of errors that have been made in the past is helpful, but keeping the functionality of such programs minimal is really the best protection available. Many existing applications have been heavily examined and tested, and many are undoubtedly trustworthy.

2. Operate the subsystem inside a restricted environment which contains the client as described in section 5.1, but leave all other files owned by the EUID/EGID outside it. This eliminates any possible accesses to those files.

3. Generate a *pseudo* user name which will own the program and its minimum necessary files.

This name would not correspond to any valid login account, which would help insure that the user id would not be used for anything but its intended purpose, and would be used only by the subsystem and its files. This fixes the modeling problem with a non-logged-in user being able to access files, and reduces the management overhead of insuring that no files are accidentally accessible via this program. A version of this could be used now in the administration of some of the protected subsystems of UNIX, such as **notes** and **uucp**, but generally is not (the special pseudo-users are in fact allowed logins, which makes auditing of the actions of users impossible if more than one user possesses the ability to login as a particular pseudo-user).

Several problems could arise with a naive implementation of this limited SETUID/SETGID in a secure system. For example, audit trail information must still log the actual user identity, not the one assumed by the SETUID program. There seem to be no significant problems implementing this and the other minor changes that are needed, and the functionality is fully compatible with the majority of applications.

5.3 Use of a General Privilege Mechanism

At the TCSEC⁷ B2 and higher levels of security, it is necessary to limit the privileges that even a privileged program can exercise. That is, even a privileged program has to operate in a mode such that it can execute only privileged operations which are mandatory for it to perform its intended function. This is to reduce the damage that a Trojan horse or an error could cause. There are several possible models and implementations which can fulfill this requirement. These include capability-based methods, privilege-set methods, and probably others. A companion paper⁴ describes a privilege-set based mechanism which eliminates the need for SETUID programs as a privilege-control mechanism. That paper concentrates on the privilege aspects of the mechanism. This section describes the use of that mechanism for implementing protected subsystems. A basic premise of the mechanism is that it should be possible to fully enumerate and control all privileged operations in a secure system, but the bulk of this discussion is oriented toward its use as a SETUID/SETGID replacement. The mechanism would be used in similar ways to help construct the TCB of a least-privilege B2 system, but we will defer that discussion.

For our purposes, all privileged operations in the system can be described by tuples of the form (*subject, operation, object*). The *subject* is the user, the *operation* is the operation that the user wants to perform, and the *object* is the system object on which the operation is being performed. (The object is sometimes obscure when discussing a privilege like "DAC exempt". In such cases, the object is generally "all objects of a given class", or the kernel itself.) A (somewhat simplified) description of how this scheme can be used to control access to files in a manner similar to SETUID/SETGID follows.

We permit the dynamic creation of privileges by designating any file system object as a *privileged object*. We then create at least three privileges associated with that file, corresponding to the

UNIX *mode bits*, read, write, and execute. (As before, other operations such as searching can be overloaded onto these.) The privilege (*user, operation, file*) would have to exist, and be available to a user's process, in order for the process to perform the corresponding operation. The details of how privileges are controlled and distributed to processes are important, but are discussed in the companion paper and so will not be reintroduced here. Two important points that are important for an understanding of this approach need to be emphasized: In general, only a program which has been examined and found trustable for a specific privilege will be able to acquire and exercise the privilege, and only users who are deemed trustworthy to exercise a privilege will be able to do so — regardless of what programs they execute and the privileges those programs are trusted to use. Both these limitations take the form of enumerating the privileges available to all users and to all programs, and placing their distribution completely under administrator control.

The function of SETUID/SETGID when used to build protected subsystems is to insure that only a specific set of users (those with access to the SETUID/SETGID programs) can execute the subsystem, and through it access the data files it protects. In our model, the data files protected by a subsystem are designated as *privileged objects*, and any operation on the files require the corresponding privilege. For example, the UNIX password and authentication file */etc/passwd* would be a privileged object. The operations of *reading the password file* and *writing the password file* would then be privileged operations, and only processes able to exercise those privileges could read or write, respectively, the password file. So, the UNIX *login* process could possess the privilege to read the password file, but not the privilege to write it, whereas a password updating program would possess both.

Figure 5 shows an example of two privileged files (*/etc/passwd* and */dev/kmem*) which are accessed by two privileged programs

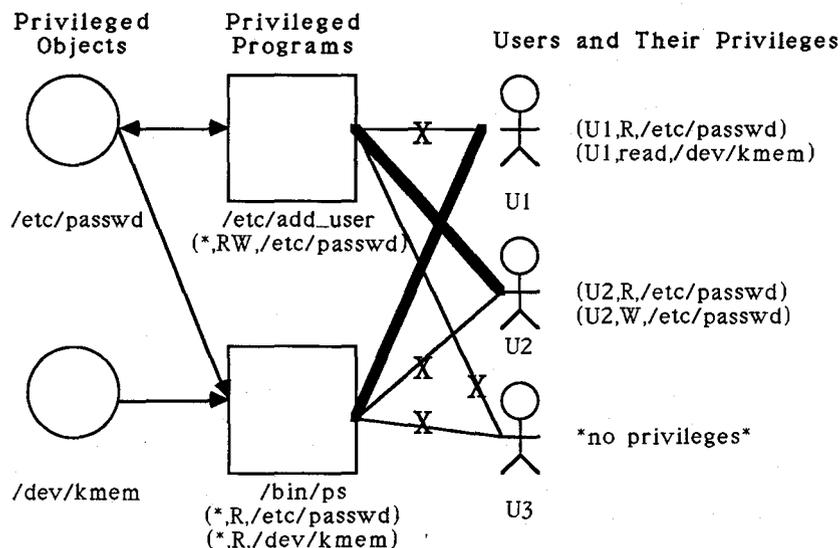


Fig. 5 - An Example of Privileged Objects and Programs (see text)

`/etc/add_user` and `/bin/ps`. Three users, U1, U2, and U3, possess various privileges. The figure shows that U1 possesses the necessary privileges for `/bin/ps`, but not for `/etc/add_user`. An attempt by U1 to execute the `/etc/add_user` program itself might be successful, depending on access controls on the program file, but the program would be unable to modify the password file. U2 possesses sufficient privilege for `/etc/add_user` to operate, but not for `/bin/ps`. U3 possesses no privileges and cannot successfully execute either program. Note that the ability to *execute* either program is totally independent of the ability of that program to affect the corresponding privileged object(s). Administrators need not worry about the discretionary access permissions on *any* of the files or programs in his system in order to control the upper bound of privileged operations which users can do. Similarly, the administrator need not worry about any program which does not possess the necessary privileges being able to access the privileged objects.

One significant difference between this method and SETUID/SETGID is the ownership and privileges attached to files created by the privileged programs. In our privilege model, programs run with UIDs equal to the executing user, so any files created belong to the executing user. With SETUID/SETGID programs, the program owner is the owner of any files created by the program. This is an important difference for programs which create files which they wish to exclusively control. (In our model, the files would be owned by the RUID, and thus the program would be unable to protect them from the user with discretionary permissions.) Further, since privileges are tightly controlled, we would not want such applications to be able to create new privileged objects lightly, so new files created by the program would not normally be privileged and would therefore not be under the control of the privilege mechanism. To solve this problem, one simply makes the directory containing the protected files the privileged object, rather than attaching privileges to each protected file, and creates new files within that directory.

The above solution makes it possible for the subsystem to create new files which are also protected by the privilege mechanism, without needing the ability to create privileged objects itself. This is a potentially visible difference for existing SETUID/SETGID applications, since some may require the ability to create new privileged objects; however, those applications which already create their new files in a private directory can remain unchanged. Because most subsystems want to control the ability of users to delete files, such private directories are frequently the case now. A private directory is made necessary by UNIX's discretionary access mechanism in order to guarantee that the controlled files are controlled solely by the application. Consequently, the set of unchanged programs includes most data base programs, mail programs, bulletin board systems, and games. It appears that this will present a small incompatibility in practice.

The privilege mechanism we have described is separate from any other discretionary or mandatory access control mechanism, and, like SETUID/SETGID, is externally applied to programs and files. If access to *any* object in the system needs to be controlled, and the other access control mechanisms are too general or otherwise inappropriate, this mechanism can provide the fine-grained control needed. This mechanism also reduces the ability of DAC-exempt privileged users or programs to violate the integrity of privileged objects. If a privileged user needs the ability to override the privilege mechanism for a particular privileged object, he can be given the appropriate privilege without compromising any other privileged objects.

Interactions with the other access control mechanisms are

straightforward to analyze, as this mechanism used in this way simply provides a *restriction* of access, and cannot allow access to take place when it could not have before. (Though the privilege mechanism in general will not have that property, as it will be used to control kernel privileges such as "DAC exempt"). Because it does not change the effective user/group of a process like SETUID/SETGID, the DAC implications are simple: the program has exactly those discretionary access rights of its executor, plus the additional capability to access appropriate privileged objects.

6 CONCLUSIONS

This paper has discussed a number of properties of the UNIX SETUID/SETGID mechanism, concentrating on security implications. Though the SETUID/SETGID mechanism is simple and powerful, it is easily misused and abused, and appears to be inadequate to alone provide the fine degree of control over privileges and protected subsystems that is required in a highly secure system.

This paper has discussed three mechanisms which replace the UNIX SETUID/SETGID with more controllable mechanisms. The first was used in Gould's original secure UNIX product, UTX/32S 1.0; it is highly secure, but is non-transparent to applications. The second is an augmentation of the first, in which a limited SETUID/SETGID facility is provided for trusted subsystems. This limited use of SETUID/SETGID appears to be compatible with the requirements of the TCSEC⁷, at least at levels of B1 and below, and will appear in future Gould systems at C2 and B1 if our implementation is successfully evaluated by the NCSC at those levels. The third mechanism, a privilege-based model, represents an approach which we believe to be new.

We believe that the privilege-based mechanism we have described is a candidate for replacement of the SETUID/SETGID concepts in a secure UNIX system. It allows user-supplied privileged subsystems and applications to be created, without interacting with its protection of the TCB. We have not yet studied all the implications of this scheme on existing applications, but it appears that most SETUID/SETGID programs will work correctly without modification. We also believe that it is far superior to SETUID/SETGID for new applications. The mechanism is not incompatible with SETUID/SETGID, so technically, the two features could coexist; this may be reasonable in a less-secure environment or one with only a few key SETUID/SETGID programs which are known to be well-behaved. Gould developed this mechanism to satisfy the least-privilege requirements of the TCSEC at levels B2 and above, a goal which it appears to fulfill.

7 ACKNOWLEDGEMENTS

We would like to express our appreciation to the people who have participated in discussions about SETUID/SETGID and the other mechanisms presented in this paper and have assisted in its preparation. These include (in alphabetical order) Phil Brewer, Bill Caudle, David Fields, John Gertwagen, David Healy, Frank Knowles, Greta Miller, Tim Thomas, and Joanna Yip from Gould, and our NCSC developmental and formal evaluation team members. We would especially like to thank the members of the NCSC team for bringing the modeling issues with SETUID/SETGID to our attention.

8 REFERENCES

1. AT&T, *9B2 Computer UNIX System V Release 2.0 User's Reference Manual*. (1985)
2. Gligor, V., et. al., *On the Design and Implementation of Secure XENIX Workstations*, Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, Ca. (April 1986)
3. IEEE, *Trial-Use Standard Portable Operating System for Computer Environments*. Institute of Electrical and Electronic Engineers, Inc. (April 1986)
4. Knowles, Frank; Bunch, Steve, *A Least-Privilege Mechanism for UNIX*, Proceedings of the 10th National Computer Security Conference, Baltimore, MD. (Sep 1987)
5. McCauley, E.J.; Drongowski, P.J., *KSOS -- The Design of a Secure Operating System*, National Computer Conference. (1979)
6. Miller, Greta, et. al. *Integrity Mechanisms in a Secure Unix: Gould's UTX/92S*. AAIA/ASIS/DODCI 2nd Aerospace Computer Security Conference, A Collection of Technical Papers. (Dec 1986)
7. National Computer Security Center, Office of Standards and Products. *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, National Computer Security Center, Fort Meade, MD. (Dec 1985)
8. NCSC Developmental Team, *Verbal Communication*.
9. Schroeder, M. D. *Cooperation of Mutually Suspicious Subsystems in a Computer Utility*, Report #MAC TR-104, Project MAC, MIT. (1972)
10. Wood, Patrick; Kochan, Stephen, *UNIX System Security*, Hayden Press. (1985)
11. University of California at Berkeley, *UNIX User's Reference Manual, 4.3 Berkeley Software Distribution*. (April 1986)

Networking of Secure Xenix† Systems

Wilhelm Burger

IBM Corporation Federal Systems Division
708 Quince Orchard Road
Gaithersburg, MD 20878

Abstract

This paper describes design and implementation aspects of a network of Secure Xenix systems. With the advent of secure systems the need arises to inter-connect these systems in a secure manner. An immediate goal is the interconnection of Secure Xenix systems with a local area net. The Trusted Computer System Evaluation Criteria [1] and the DNSIX secure network architecture [2] are used for deriving additional security requirements in the areas of security policy and accountability to extend B2 functionality to a network of Secure Xenix systems [3]. Part 1 of the paper extends the security requirements to the network environment, part 2 describes the network security design, and part 3 addresses some implementation issues.

1. Network Security

The security mechanism of one system must be extensible to another system on the network with which it communicates. This results in a network of systems which are governed by a single security policy. In addition to mandatory and discretionary access rules, the relations between systems, their security level, the security level of their interconnection, and the authorization level of remote users and applications must be taken into consideration when objects are accessed.

The interaction between two systems is organized into sessions. Sessions are used to mediate all network access. A session consists of a set of related communications. For the duration of a session, subjects and objects on both systems are covered by the same security access rules. A session is, therefore, associated with a security level to accomplish this. The session security level is derived from the security level with which the user logged in. A session is further associated with an identification which is used for auditing security relevant network events.

All packets sent between systems are further associated with a security label. The security label is derived from the security level of the session. In a network of systems with different security levels this label is also used to route packets according to the security level of intermediate systems and links.

In the area of accountability additional network related identification and authorization data must be maintained. These 'network profiles' are used to identify and authenticate remote users; they are further used to authorize the establishment of sessions.

It is assumed that the local area network under consideration is physically secure. Otherwise additional measures are needed to deal with data compromise and data integrity. The risk of data compromise can be reduced through link encryption. Data integrity can be improved through additional encrypted check fields in data packets. In a very hostile environment stronger measures such as application level end-to-end encryption and notarization of packets will be required.

Another threat to be considered is denial of service. Denial of service is security relevant if, for example, audit service or authentication service is affected. Unfortunately, it is rather difficult to specify general criteria what constitutes denial of service. The risk of denial of service due to excessive traffic on other sessions can be minimized by giving each session a fair share of the network resource.

2. Network Security Design

The system architecture of Secure Xenix serves as the basis of the secure network extensions. The trusted computing base of Secure Xenix is enhanced with a session mechanism and with trusted agents which handle all security related communication between systems. The network security facilities rely upon fundamental features of Secure Xenix: access mediation to protected resources by the kernel, enforcement of mandatory and discretionary security policy, and the protection of authentication and audit data. Additionally, the trusted facility management of Secure Xenix is augmented with network security management functions.

All communications originating outside the trusted computing base require the establishment of sessions in order to mediate access to the network. Security related communication is handled by trusted agents. An agent usually has a counterpart on the machine it communicates with, and a client/server relationship exists between the two. A simple datagram based request-response protocol is sufficient for the communication between trusted agents. A minimal set of trusted agents consists of a session setup service, an audit service, and a network management service.

Before a user (or program on his behalf) can communicate with another system on the network a session must be established. This requires that the user is allowed to establish a session to the desired system, and that the user can be identified and authenticated at that system with his current security level. Ignoring authentication issues for the moment a session is established as

† Xenix is a trademark of Microsoft Corporation.

follows. The session setup service communicates the identity and security level of the user from the source system to the target system. At the target system a session server process is created. This process acts as proxy for the user; it has the user's identity and security level. Additionally, session control structures are created; they are needed for relating connections to sessions. Session creation can be combined with the invocation of a server process for an application, such as is needed for a file transfer program.

Network access can be controlled in a centralized or in a distributed fashion. When control is centralized then all network access profiles are stored at one place on the network. When control is distributed then each system of the network has its own network access profile. For the centralized case an authentication service comes into play which authorizes a session creation. In the distributed case session setup can be combined with network authentication, and no authentication service is needed. Network access profiles which are maintained locally have the

The secure network administrator function further has to interact with the secure administrator function of Secure Xenix to handle the registration of remote users and the mapping of their user and group identifications from one system to another.

Additional security support services can be combined with the session mechanism such as an encryption key service. If application-to-application encryption is desired then encryption keys would be obtained at session setup. All packets communicated by that session would then be encrypted and decrypted using these keys.

3. Implementation Aspects

The implementation of the secure network facilities for Secure Xenix is guided by these objectives: (1) no modifications to Secure Xenix proper are permitted, (2) the network software must be structured according to B2 requirements, and (3) the software interfaces should be adaptable to support different protocols and network interface requirements.

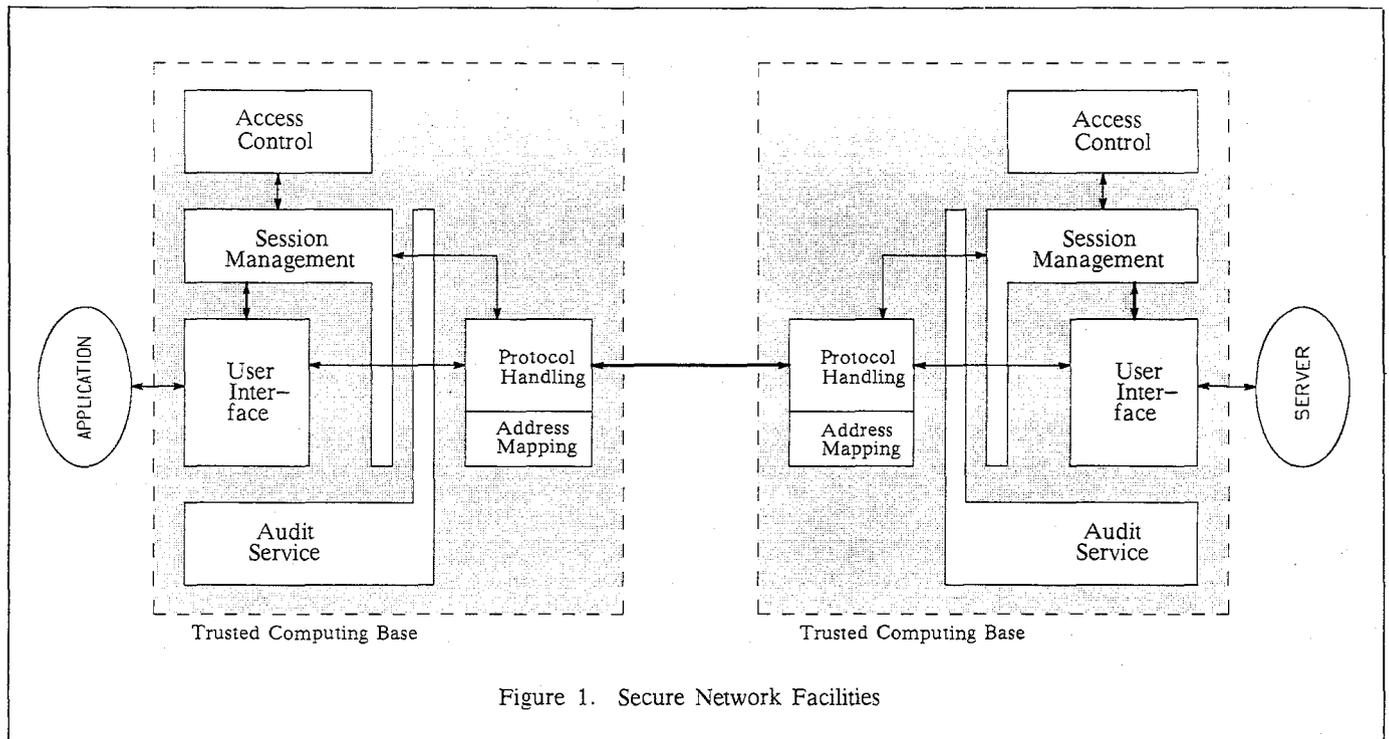


Figure 1. Secure Network Facilities

advantage that each system administrator has control over who can access the system from a remote location.

For audit purposes a session is associated with a unique identification. Audit records are generated, for example, for session establishment and session termination. Audit records are first collected at the site where they are created, and then they are sent to the audit site. This mechanism deals with the potential failure of the audit site. An audit service transfers periodically, or at the request of the auditor, audit information to the audit site. The audit collection site may be changed from one system to another by the network auditor.

The network auditor is one example of the enhanced functions of the trusted facility management of Secure Xenix for a secure network. Trusted programs and services are also required for a secure network operator and for a secure network administrator.

The first objective is needed to prevent network changes from invalidating the security rating of Secure Xenix. This objective is met by placing all security-relevant network code into device drivers and trusted processes. It allows also assurance and documentation for the network security features to be provided in an incremental fashion.

Software which belongs to the trusted computing base must be organized and structured in such a way that it cannot be by-passed or modified in any unauthorized manner. Network interfaces which would provide security relevant functions with user libraries, therefore, cannot be used as no guarantee can be given that an application uses the proper library or does not modify the code. It is also not feasible to include applications with network needs into the trusted computing base as this would substantially increase the assurance burden. Thus, the network interfaces must be protected by the Secure Xenix kernel. As the

addition of system calls to the kernel is not permitted, all network interfaces must be implemented with pseudo device drivers.

The network interface of Secure Xenix should also be adaptable to different network requirements. Protocol independence of the user visible interface is achieved with the socket mechanism [4]. Sockets provide stream or datagram services between applications, independent of the underlying protocols used. A more difficult problem is to also support network interconnections where the protocols are handled in a front-end processor. The best achievable here is the placement of all protocol processing functions into a network daemon. That way appropriate interfaces and software structures are provided which can be re-used for a host/front-end protocol.

The extensions of the trusted computing base of Secure Xenix and the structure of the secure network facilities are shown in Figure 1. All protocol handling is done by the network daemon; this daemon also manages the address mapping between network addresses and Ethernet addresses. Packets received from the user interface are wrapped with the appropriate protocol headers, affixed with a security label, and sent to the network. Packets received from the network are unwrapped of their protocol headers, checked for their destination's security label, and moved to the destination's user interface.

Session management and audit service are also implemented as daemons. Session management handles session setup, network access authentication, and session termination. Connection establishment is also under the control of the session mechanism. The audit service audits session and connection setup and termination. It also handles the transfer of audit information to a location of the network auditor's choosing. Not shown is the network management daemon. It allows for the inspection of the session and socket control structures, and for the termination of connections and sessions.

The session mechanism requires that all related connections are under the control of a session. Only processes within the same process group which created the session, therefore, are allowed to establish connections belonging to that session. To enforce this a file handle associated with the open session is used as key when creating the endpoint of a connection belonging to that session. The file handle is inherited by all children of the process group thus allowing the establishment of additional connections. As a session can be opened only once this assures that no process outside the process group can create a connection.

The actual implementation of the secure network features is carried out for an Ethernet based network using the TCP/IP protocol suite [5]. Basic mechanisms such as sessions, sockets, and the network daemon are in place, though at present only the UDP protocol is supported. A major effort is still required to provide assurance for the network security features.

References

- [1] Department of Defense - Computer Security Center, *Trusted Computer Systems Evaluation Criteria*, CSC-STD-001-83, August 1983.
- [2] Defense Intelligence Agency, *DODIIS Network Security Architecture and DODIIS Network Security for Information Exchange (DNSIX)*, DRS-2600-5466-86, May 1986
- [3] Gligor, V. et al., *On the Design and Implementation of Secure Xenix Workstations*, IEEE Transactions of Software Engineering, Vol SE-13, 2, February 1987
- [4] Leffler, S. et al., *4.2BSD Networking Implementation Notes*, Computer Systems Research Group, U.C. Berkeley, July 1983
- [5] SRI International, *Internet Protocol Transition Workbook*, March 1982

Disclaimer: The work described herein is part of a research project. No IBM product commitment is made or implied.

A LEAST PRIVILEGE MECHANISM FOR UNIX®

Frank Knowles
Steve Bunch

Gould Computer Systems Division
1101 East University Avenue
Urbana, IL 61801
217-384-8500
knowles@gswd-vms.arpa
ihnp4!uiucdcs!ccvax!knowles
srb@gswd-vms.arpa
ihnp4!uiucdcs!ccvax!srb

INTRODUCTION

This paper describes a privilege control mechanism for the UNIX® operating system. The mechanism is intended to satisfy the B2 requirement of *least privilege*⁶, and to provide fine-grained control over access by users to services and objects. The mechanism is largely independent of other security-related features and is useful as an incremental addition to a less-secure UNIX.

The principal features of this mechanism are:

- Separate privilege sets to manage the inheritance of privilege as distinct from the exercise of privilege.
- Decomposition of the UNIX *super-user* privilege into distinct privileges.
- Discretionary privileges that can be assigned at the command level.
- A replacement for the UNIX *setuid* feature which is compatible with it and can exist side-by-side with it.
- Extensible set of privileges. Users can create trusted applications which use new privileges (However, we don't discuss this aspect in this paper.)
- Compatibility with standard UNIX. The mechanism is externally applied, so applications from a non-secure UNIX, including most *setuid* applications, can run without being recompiled.

Privilege sets assigned to users, or program files, or both is not a new idea. It is derived from the capability research of the 1970's. For instance, file privilege sets were used in KSOS-II⁴, and both kinds of sets are used in Digital Equipment Corporation's VMS®. Also, assigning privileges by command (discretionary privileges) is a feature (*wheel* and *operator* concepts) of BBN's TENEX system.

The novel feature of our privilege mechanism is the first item in the bullet list above. We will define several privilege sets which will interact to do two things: impose a strict inheritance of *possible* privileges on a process hierarchy, and allow selective *activation* of those privileges when a program file is executed or a privilege is assigned at command level.

The paper is organized as follows. The following section is an overview of the privilege mechanism. The remaining sections provide details, examples, and implementation notes. The sections describing the privilege sets themselves are the heart of the paper. There is a section summarizing the privilege set recomputations as done by key system calls such as *fork* and *exec*, and a section with security policy statements regarding privilege sets.

OVERVIEW

The privilege control mechanisms of standard UNIX are the *setuid/setgid* and *super-user* features. A companion paper¹ discusses the security aspects of *setuid* and *setgid*. The B2 requirement of *least-privilege*⁶ demands a granularity of privilege finer than *no privilege* and *all privilege*. Though the UNIX *super-user* privilege could be represented in a B2-rated system as a single privilege implying all other privileges, a range of privileges is also required.

Common methods for conferring privilege are: associating privilege sets with users, associating privilege sets with program files, and doing both. Associating privileges with program files is done in standard UNIX (with *set/ug/id* bits on a file) and in a number of other systems including KSOS-11⁴ and VMS. Associating a privilege with an executable file allows a user access to privileged operations in a restricted way -- the program that uses the privileges is distributed with the system and cannot be controlled by the user. Other protection mechanisms such as file protection modes or login environments⁵ are used to restrict access to the privileged executable programs. (A disadvantage, often seen in practice, is that the other mechanisms (file modes or access lists) are relied upon for the integrity of the executable files, rendering the privilege mechanism an extension rather than an independent addition to the overall protection scheme. Our privilege mechanism is easily implemented with a enough privileges to protect the privilege database -- for instance, it would be a privilege to execute a file with a non-empty privilege set.)

A central problem with associating privileges with program files is the problem of controlling propagation of privileges. In standard Unix, it is possible for a non-privileged user to acquire privilege by executing a *setuid* file. In UNIX and other systems which associate privileges with users as well as with files, a user may acquire a privilege not in the user privilege set by executing a file with that privilege. Our mechanism controls the propagation of privilege in two ways: strict inheritance of *possible* privileges, and dynamic recomputation of *usable* privileges. At login time, the login process for a user is given a set of privileges (a bounding set) associated with the user that contains all privileges that could possibly be exercised by processes in the process hierarchy determined by this process. However, if these privileges were also enabled at the time they were conferred, the initial process (and possibly many others) would be greatly over-privileged. This would not satisfy the B2 requirement of *least-privilege*. We solve this problem by interpreting this bounding set as the set held "in trust" for descendants. The privileges cannot be executed just because they are in the bounding set -- something not completely under the control of the process must enable them. Privileges are enabled when a program file is executed. A particular privilege becomes available to a process when the privilege is in both the bounding

set of the process that executed the file and in the privilege set associated with the file. Thus, a program file enables a privilege already known to the user rather than granting a privilege new to the user.

The connection just described between a bounding set that limits a process hierarchy (descended from a login process) and a file privilege set associated with a program file is the novel feature (and the most important feature) of the privilege mechanism described in this paper. As will be seen, this mechanism requires several distinct process privilege sets (recomputed by `exec` when a program file is executed) in addition to relatively static privilege sets associated with users and program files.

An additional feature is the ability of a user to assign a privilege to a particular process executing a program file rather than to the file itself. (Privileges that can be assigned in this way -- discretionary privileges -- must also be in the process bounding set.) This feature is similar to the *wheel* and *operator* concepts mentioned earlier. This concept is useful for security testing, or for use in a less-secure environment, or for use where the UNIX super-user feature is desired. Discretionary privileges were added to the basic model at the request of our Real-Time UNIX group. They wanted the privilege mechanism to control access to certain real-time services which can take over or even crash a system and preferred to trust the system programmers to use the privileges with prudence during development. This attitude is typical in development environments.

The remainder of this paper describes our proposed implementation of this scheme in detail.

PRELIMINARY DEFINITIONS

Popular usage of the term *privilege* is sometimes confusing. One definition is that a *privilege* is an operation which, if misused, could violate the security policy of the system. Thus, "writing the password file" is a privilege. Another definition is that a privilege is the *capability* to perform an operation which could violate the security policy of a system. Either notion will work. We adopt the latter view in this paper and define a *privilege* as a relation whose members are tuples of the form: (subject, mode, object). Thus one privilege might be (jones, write, /etc/passwd). We say very little in this paper about the encoding of privileges. It is sufficient to assume that privileges are mapped in some way onto integers. It may even be advisable in an implementation to map mode and object pairs separately onto integers to facilitate the assignment of subjects to mode-object pairs. Another concern is that the concept of mode is not a simple one. Some privileges cause the disabling of particular security or integrity checks rather than enabling a particular action. Overriding of discretionary access controls is an example of this. Privileged operations will include operations such as:

- override mandatory access checks (security labels⁶)
- override discretionary access checks (file modes or access lists)
- use a particular system call
- connect to a particular port
- execute a particular file
- write a particular file
- execute I/O instructions directly (a real-time extension to UTX/32)

The list above is meant to be representative rather than exhaustive.

There is an increasingly well-known jargon associated with secure computing systems. We will use the terms, *trusted software*, and *security policy*, as defined in the *Department of Defense Trusted Computer System Evaluation Criteria*⁶. For the reader's convenience, we define them again here (though somewhat rephrased). The *security policy* of a system is a statement of the rules which the system enforces in order to both protect sensitive information stored in the system and to protect the system itself from penetration or unauthorized modification. *Trusted software* is software that is relied upon to enforce the security policy.

One concept from standard UNIX is closely connected with privilege-granting. In UNIX, there is a *real user id* associated with each process. It is an identifier that denotes the user on whose behalf the process is operating. How closely this is really the case varies with the version of UNIX being used. In Gould's secure UNIX, UTX/32S[®], an extra id, the *log id* is used to unequivocally identify the user responsible for a process. However, in this paper, we wish to avoid using concepts specific to UTX/32S, so we will be content with the real user id.

THE BOUNDING SET DATA BASE

In this section we introduce the terminology of privilege sets and briefly indicate their interconnections. There are two kinds of privilege sets: those associated with relatively static system objects and those associated with processes. We mention both kinds, however, those associated with solely with processes will be explained in later sections.

A *user bounding set* is a set of privileges associated with a user entry in the system password file. At login time, this set is associated with the user's login process as the *process bounding set* of the process. The process bounding set of a process contains all of the privileges that can possibly be exercised by that process or any descendent of that process. Having a privilege in the process bounding set does not mean that the privilege can, in fact, be used. This will be explained in detail later. Another set associated with a user entry in the password file is the *discretionary set*. At login time, a copy of the user's discretionary set becomes the discretionary set of the process. The discretionary set is a subset of the user bounding set and contains those privileges that can be assigned to a particular invocation of a program file, as distinguished from being assigned to the file itself.

Each program file has a *file bounding set* consisting of those privileges that the program is trusted to exercise. (In practice, most program files will have empty file bounding sets.) When a program file is executed, those privileges in the process bounding set that are also in the file bounding set (or the discretionary set) are made *active* and thus capable of being used. This activation is done by using two other privilege sets that will be defined later -- *potential sets*, and *active sets*. We will refer to sets that contain privileges, both those defined in this section and those defined later, as *privilege sets*.

PROCESS BOUNDING SETS

A *process bounding set* is associated with each process. The process bounding set contains all of the privileges that a process and its descendents may ever use. It is computed when the process is created and is recomputed if the real user id of the process is changed. The process bounding set of a parentless process (e.g., the `init` process) is set by the system initialization code. After system initialization, no process may add a privilege to its own process bounding set or that of any other process, but any process may delete a privilege from its own process bounding set. A login process takes as its process bounding set the user bounding set of the user. A child process created by `fork` inherits its process bounding set from its parent. A new process image

formed by `exec` inherits its process bounding set from the calling process. If a process has deleted members from its process bounding set, any descendent of that process inherits the diminished set, not the original set. Consequently, as one descends the process hierarchy, the process bounding set may get smaller but cannot get larger.

Any system call that changes the real user id of a process causes a recomputation of the process bounding set. The new process bounding set is the intersection of the old process bounding set and the user bounding set of the new real user. A (*trusted*) system call that changes the real user id of a process must pass the user bounding set of the new user to the kernel so that privilege set recomputation can take place. Note that even though user bounding sets can change during the lifetime of a process, a process cannot gain privileges from a recomputation of its process bounding set because the new set is always a subset of the previous set.

DISCRETIONARY SETS

The reason for having discretionary privileges is to designate those privileges that can be assigned to a process when it executes a program file -- as distinguished from assigning the privilege to the file itself. Though the program file can be given privileges directly by writing its file bounding set, some privileges are so awesome -- or must be turned off and turned on so often -- that it is best to require their attachment only at the moment of use. Also, in some environments, system programmers are trusted to use prudence in their use of privileges and generally suffer the consequences themselves if they make a mistake. This is the case, for instance, when debugging a program that exercises *direct I/O* (a real-time extension to Gould's standard UTX/32.)

Each user entry in the system password file has associated with it a *discretionary set* of privileges. It is a subset of the user bounding set in the same entry. A new login process inherits a copy of this set as its own discretionary set. Each privilege in the process' copy of the discretionary set has a flag that indicates if the privilege is *turned on* or *turned off*. A process may *turn on* or *turn off* any of its discretionary privileges at any time. A process with a full set of privileges in its bounding and discretionary sets and all discretionary privileges *turned on*, could bestow the UNIX *super-user* privilege to its children.

The discretionary set of a parentless process is set by the initialization code. When a process is created by `fork` it inherits its discretionary set from its parent process. The new process image formed by `exec` has the same discretionary set as that of the calling process, however, `exec` *turns off* all discretionary privileges in the new discretionary set. (This is to avoid accidentally passing turned-on privileges to second-generation descendents. First-generation descendents can be passed immediately usable discretionary privileges, as described later.) If the real user id of a process is changed, the discretionary set is set to the intersection of the process bounding set and the discretionary set associated with the new user id (again with all privileges turned off).

We will illustrate this mechanism with an example. In this example, `priv` is a (new) command internal to the shell that exists just for the purpose of processing discretionary privileges. A program `mass_write` does a massive rewrite of a disk volume and for increased performance executes hardware I/O instructions directly. Any process which does this must have the privilege, `direct_io`. A user that has `direct_io` in his or her bounding set could, by typing the following command, execute `mass_write` and temporarily give it the `direct_io` privilege:

```
priv direct_io mass_write
```

`Priv` executes a system call that *turns on* the discretionary privilege `direct_io`. `Direct_io` must, of course, already be a member of the discretionary set of the shell process that the user is communicating with, and this implies that `direct_io` is in the discretionary set of the user. When `exec` is called, it does two things with regard to the discretionary set. It adds each *turned-on* discretionary privilege to the *potential set* (defined later) of the new process, and it passes the discretionary set of the calling process to the new process image after *turning off* each privilege. That the new process has the `direct_io` privilege available to it when it begins execution and need not enable it explicitly follows from the discussion in the next two sections on *potential sets* and *active sets*.

POTENTIAL SETS

Each process has associated with it a *potential set* derived from its process bounding set, its discretionary set, and the file bounding set of the program being executed. The potential set contains the privileges that can be *exercised* by the process, as distinct from the privileges that can be *passed on* by the process -- the latter are defined by the process bounding set. The potential set represents the combination of the user and program file privileges that the executing process may actually use. A process may delete a member from its potential set, but may not add a member. The usefulness of the potential set is apparent when subprocesses with few or no privileges (such as a login shell) are interposed between processes that can exercise privileges.

The potential set of a parentless process is specified by the system initialization code. The potential set of a new process created by `fork` is the same as the potential set of the parent process. A new login process takes as its potential set the file bounding set of the `login` program -- this will be illustrated later in an example. When the real user id of a process is changed, potential set is recomputed as the the intersection of the old potential set and the recomputed process bounding set. Thus, the potential set is always a subset of the process bounding set.

The potential set of a new process image created by `exec` is the intersection of the process bounding set of the process calling `exec` (which is the same as the new process image's bounding set) and the set formed by the union of the file bounding set of the file to be executed and that subset of the discretionary set consisting of those privileges that are *turned-on*. Note that the new potential set is a subset of the new process bounding set. This recomputation does three things: brings in the privileges of the file to be executed, allows discretionary privileges to be added, and keeps the potential privileges within the bounding privileges. This computation is an extremely important part of the privilege mechanism because it makes it possible for the privileges kept in trust in the process bounding set to become available for use.

ACTIVE SETS

The privilege sets discussed up to this point manage the propagation of privilege to new processes. The process bounding set is an absolute bound for a process and all its descendents; the potential set bounds only the process itself; and discretionary privileges are sweeteners that can be added along the way. At any point in time, the *active set*, described in this section, contains precisely those privileges that are active, i.e., can be exercised. It is always a subset of the potential set. It is this set that trusted code uses to decide what privileges are available when a privileged operation is requested.

The active set facilitates the localization of privilege usage in trusted code. Specifically, a process may eliminate all privileges from its active set before performing a series of unprivileged actions, then re-insert privileges from its potential set into its active set as they are needed. This feature reduces the effort of

code inspection for trusted processes, and helps satisfy the least-privilege requirement in a B2-rated system⁶.

The active set of a parentless process is specified by the system initialization code. When a new process is formed by a `fork`, the active set is inherited from the parent process, but any process may change its active set, subject to the general rule that the active set is always a subset of the potential set. When the real user id of a process is changed, the active set of the process must be recomputed since the potential set is recomputed. The new active set is the intersection of the previous active set and the new potential set. When a new process image is created by `exec`, the active set is initialized to the new potential set as recomputed by `exec`. (Thus we see that when a discretionary privilege is added to a new potential set by `exec` (as described in the section on discretionary privileges), it is also added to the new active set, and thereby becomes a privilege that can be exercised.)

PRIVILEGE COMPUTATION SUMMARY

In the previous sections, each privilege set was discussed individually. In this section, we summarize how these sets are computed.

LOGIN

The process bounding set of a new login process is set equal to the user's bounding set in the password file. The discretionary set of the process is a copy of the user's discretionary set (also in the password file). The potential set and the active set are set equal to the file bounding set of the program file `login`. The potential and active sets will be recomputed when the login shell is `exec-ed`. This will be clarified later by an example.

FORK

`Fork` doesn't change any process privilege set. The process bounding set, the discretionary set (and their *on/off* state), the potential set, and the active set of the new process are identical to those of the parent process.

CHANGE REAL USER ID

There may be several system calls that change the real user id of a process. The rule, stated below, applies when any of these system calls is used.

The new process bounding set is the intersection of the old process bounding set and the user bounding set associated with the new real user id.

The new discretionary set is the intersection of the new process bounding set and the discretionary set associated with the new real user id. All discretionary privileges in the new discretionary set are *turned off*.

The new potential set is the intersection of the old potential set and the new process bounding set.

The new active set is the intersection of the old active set and the new potential set.

EXEC

The new process bounding set is the same as that of the calling process.

The new discretionary set is the same as that of the calling process except that all privileges are *turned off*.

The new potential set is the intersection of the new (same as old) process bounding set and the set formed by the union of the file bounding set and the set of discretionary privileges *turned on* in the discretionary set of the *calling* process.

The new active set is the same as the new potential set.

SECURITY POLICY

This section summarizes the security policy as it applies to the acquisition and exercise of privilege. The policy has several aspects each of which is encapsulated in a separate statement:

- The kernel as well as other trusted code is trusted to deny a request by a process if the requesting process does not have the appropriate privilege.
- After system initialization is completed, no process may change a privilege set associated with another process; no process may add privileges to its own process bounding set or to its own potential set.
- If process A is a descendent of process B, then the process bounding set of A is a subset of the process bounding set of B.
- For each process, its active set is a subset of its potential set, and its potential set is a subset of its process bounding set.
- During the lifetime of a process, the only privileges that it may use are those in its potential set -- as initialized at the creation of the process.
- At any point in time, the privileges that can be exercised by a process are precisely those in the active set of the process.
- When a new process image is formed in order to execute a program file, its active set is contained in the union of the file bounding set of the file being executed and the discretionary set of the user.
- If any file is modified, its file bounding set is made empty.

EXTENDED EXAMPLE

In this section we work through the recomputations of process privilege sets that occur as a user logs onto the system and types a couple of commands. Each creation of a new process or of a new process image, each changing of the real user id, is indicated below as a separate, numbered action:

```
(1) init --fork--> process1
(2) process1 --exec--> getty
(3) getty --exec--> login
(4) login -setreuid-> process2
(5) process2 --exec--> csh
(6) csh --fork--> process3
(7) process3 --exec--> chmod
(8) csh -priv(8)-fork-> process4
(9) process4 --exec--> mass_write
```

For each numbered action, we will show the privilege sets of the resulting process. The reader should be able to check the recalculation for a particular step by starting with the previous privilege sets and applying the rule for the action taken.

Instead of defining realistic sets of privileges (and explaining them), we merely indicate a set of privileges as a set of integers, each integer being assumed to stand for some privilege, i.e., [1,2,3] is a privilege set with privileges: 1, 2, and 3. In one case, privilege 8 (see below), we do specify that it stands for the **direct_io** privilege just to be consistent with the example presented earlier in the section on discretionary sets. When a privilege in a discretionary set is *turned on*, it will have an "*" beside it.

We now state the assumptions that hold before the first action takes place. We assume that the process **init** has the following privilege sets: process bounding set = potential set = active set = [1,2,3,4,5,6,7,8], and an empty discretionary set (remember, each number corresponds to a privilege, and privilege 8 is **direct_io**). We assume that the program file **getty** has file bounding set = [2,3,7], and the program file **login** has file bounding set = [2,3,5,7]. We assume that the **setreuid** system call changes the real user to that of a user whose bounding set = [3,4,7,8] and whose discretionary set = [7,8]. We assume that the program file **csch** has an empty file bounding set. Finally, we assume that the **chmod** program file and the **mass_write** program file each have a file bounding set = [3,4,5].

The recomputed privilege sets after each action are shown below.

(1)
 process bounding set = [1,2,3,4,5,6,7,8]
 discretionary set = [empty]
 potential set = [1,2,3,4,5,6,7,8]
 active set = [1,2,3,4,5,6,7,8]

(2)
 process bounding set = [1,2,3,4,5,6,7,8]
 discretionary set = [empty]
 potential set = [2,3,7]
 active set = [2,3,7]

(3)
 process bounding set = [1,2,3,4,5,6,7,8]
 discretionary set = [empty]
 potential set = [2,3,5,7]
 active set = [2,3,5,7]

(4)
 process bounding set = [3,4,7,8]
 discretionary set = [7,8]
 potential set = [3,7]
 active set = [3,7]

(5)
 process bounding set = [3,4,7,8]
 discretionary set = [7,8]
 potential set = [empty]
 active set = [empty]

(6)
 process bounding set = [3,4,7,8]
 discretionary set = [7,8]
 potential set = [empty]
 active set = [empty]

(7)
 process bounding set = [3,4,7,8]
 discretionary set = [7,8]
 potential set = [3,4]
 active set = [3,4]

(8)
 process bounding set = [3,4,7,8]
 discretionary set = [7,8*]
 potential set = [empty]
 active set = [empty]

(9)
 process bounding set = [3,4,7,8]
 discretionary set = [7,8]
 potential set = [3,4,8]
 active set = [3,4,8]

When **mass_write** exits, the parent shell process (**csch**) has privilege sets as indicated in step (5).

ADD USER EXAMPLE

In this section we will consider a simpler scenario than that of the preceding section. Our concern is not with how privileges are computed but rather with how they can be assigned. The application we are interested in is the administrative chore of adding a new user to the system. This example is taken UTX/32S, but this interface to the authentication database could easily be implemented in any standard Unix. An administrator executes a program file **add_user** which is a client for a *trusted* server. That is, the client process connects to a known socket where a server *daemon* is listening for requests. The daemon authenticates the user and **forks** a process that **execs** a program file **au_back_end** that is not directly accessible to users. The new process image is the *dedicated* server that actually carries out the user request. The administrator communicates directly with the dedicated server and using a menu-like interface requests certain changes to the password file. In this example, a new user entry would be added to **/etc/passwd**.

Notice that each separate action -- execute a client, connect to a socket, execute a back-end program file, request a specific change in the password file -- are candidates for separate privileges that can be assigned to different user bounding sets or file bounding sets in a way deemed the best compromise between providing security and being user-friendly. Only those administrators trusted to add new users would have the privilege in their user bounding set to execute the client **add_user**. Only the file **add_user** would have the privilege in its file bounding set to connect to the applicable socket. The server daemon would need no privilege beyond that of **binding** to the right socket. The file **au_back_end** would have the privilege in its file bounding set to modify the password file. The various tasks involved in maintaining the password file could be separate privileges in order to enforce the *two-person rule*. For instance, there could be a privilege for *entering* a new user entry and a privilege for *activating* a user entry. No administrator would possess both privileges.

It is interesting to note that a malicious user not privileged to write privilege sets would be unable to write a Trojan-horse version of **add_user** capable of affecting the password file. Such a program -- even if executed by an administrator authorized to add users -- would not be able to connect to the right server socket or be able to write the password file directly because the needed privilege would not be in the file bounding set of the program file!

IMPLEMENTATION NOTES

In this section we discuss some implementation issues.

A surprisingly small number of system calls is needed to implement the privilege mechanism. These calls are described below. Access checks are not fully delineated, so keep in mind the restrictions as set forth in previous sections. The system call arguments used below -- b, d, p, a, f -- represent the the process bounding set, the discretionary set, the potential set, the active set, and the file bounding set, respectively. Also, the structure <priv> is a privilege set with an on/off flag for each of its members.

There are two system calls for manipulating the privilege sets associated with a process or program file.

```
<priv> = read_pset( [b d p a f] | [file] )
```

Read one of the privilege sets of this process (file).

```
write_pset( <priv>, [b d p a f] | [file] )
```

Sets one of the privilege sets of this process (file). This call can be used to delete members from the process bounding set, the discretionary set, the potential set, and the active set. It can turn a discretionary privilege *on* or *off*. It can add members to the active set as long as the active set remains a subset of the potential set. These are all unprivileged operations. Modifying a file bounding set, however, is a privileged operation.

Inside the operating system kernel, many operations may become privileged operations in a secure system. In order for the kernel to determine if an operation is privileged and, if so, if the caller is privileged to perform that operation, an additional system call is needed:

```
<true/false> = has_priv( <subject id>, <access mode>, <priv-id> )
```

Determine if this subject may access this object in this mode. The call fails if a privilege is needed and the subject doesn't have it.

System calls that change the real user id of a process must pass the applicable user bounding set to the kernel so that privilege recomputation can be done. The reason for this is that user bounding sets will likely be stored in a file instead of in a kernel table. Of course, the system calls in question are privileged, and the calling processes are trusted to pass the correct information.

As described in the section on discretionary sets, a command internal to the shell (**priv**) is desirable in order to enable a discretionary privilege for a new process.

Implementing privilege sets as bit vectors is straightforward except perhaps for the question of where file bounding sets are to be stored. Assume for the moment, that file bounding sets are attached to inodes in the file system. It would be desirable at system startup to verify that the file bounding sets on the disks are set correctly (e.g., that there are no new or modified privileged programs). This can be done by shell scripts which verify the contents of bounding sets for specific files, check for the existence of unauthorized non-empty bounding sets, and compute a cryptographic checksum on files with non-empty bounding sets. On a small system, this could be done by an exhaustive search of the file system. On a large system, this could be an intolerably slow process. An alternative method is needed. For example, one could store the names of all the privileged processes, their file bounding sets, and their file checksums in a startup file. At start-up time, a program using the start-up file would install the file bounding sets by attaching them to file inodes locked in memory. The file bounding sets would never be attached to disk

inodes. An inode not in this memory-resident cache would have an empty file bounding set.

FURTHER ISSUES

The discussion of a number of issues has been deferred to other papers. Some of these will not be fully resolved until we have experience with this mechanism in a variety of situations. These include:

- The optimal encoding of privilege vectors. This is an important aspect, affecting both performance and extensibility of the mechanism.
- The enumeration of specific privileges appropriate to UNIX.
- Extensibility to user-assigned privileges.
- Use of this mechanism in implementing security policies requiring fine-grained integrity models in addition to data security.

ACKNOWLEDGEMENTS

The ideas in this paper were developed out in a series of discussions with different people in attendance at different times. The discussions included Dave Healy, Greta Miller, Tim Thomas, and the Real-Time UNIX contingent, John Gertwagen and Scott Preece.

REFERENCES

1. Bunch, Steve. *The Setuid Feature in Unix and Security*. Proceedings of the 10th National Computer Security Conference. (Sep 1987) Baltimore, MD.
2. Hecht M. S., et. al. *UNIX Without the Superuser*. Summer USENIX Technical Conference and Exhibition. Phoenix, AZ. (Jun 87)
3. Lampson, B. W. *Protection*. Proc. Fifth Princeton Symposium on Information Sciences and Systems. (Mar 1971).
4. McCauley E. J., Drongowski P. J. *KSOS -- The Design of a Secure Operating System*. National Computer Conference. (1979).
5. Miller, Greta, et. al. *Integrity Mechanisms in a Secure Unix: Gould's UTX/32S*. AAIA/ASIS/DODCI 2nd Aerospace Computer Security Conference, A Collection of Technical Papers. (Dec 1986).
6. National Computer Security Center, Office of Standards and Products. *Department of Defense Trusted Computer System Evaluation Criteria*. DoD 5200.28-STD. (Dec 1985) Fort Meade, MD.
7. Schroeder, M. D. *Cooperation of Mutually Suspicious Subsystems in a Computer Utility*. Report #MAC TR-104, Project MAC. MIT.

AN OVERVIEW OF THE DoD COMPUTER SECURITY RDT&E PROGRAM

Panel Chairman, Mr. Lawrence Castro
Chief of the Office of Research and Development
National Computer Security Center

The purpose of this panel is to inform the audience of the progress of and plans for the Research, Development, Testing, and Evaluation (RDT&E) efforts sponsored by the Department of Defense (DoD) Computer Security Program (CSP).

The presentation is organized according to the five distinct areas of the R&D Program: Secure Architecture, Secure Database Management Systems (DBMS's), Network Security, Modeling and Verification, and Aids to Evaluation.

The first part of the presentation will allow each panel member to describe the status of his area's current programs and new initiatives for FY88. In addition, we will include a progress report of the multilevel secure workstation program. Among the new initiatives to be described are those related to support of the Strategic Defense Initiative (SDI). The participating panel members from the three military service labs will describe the support they are providing to the CSP. Following this, the panel will entertain questions from the floor.

Panel Members:

CDR David Vaurio, Deputy Chief,
Office of Research and Development
(R&D), National Computer Security
Center (NCSC)

Mr. Wayne Weingaertner, Office of
R&D, NCSC, Secure Architectures

Dr. John Campbell, Office of R&D,
NCSC, Secure DBMS

Mr. George Stephens, Office of R&D,
NCSC, Network Security

Mr. Rob Johnson, Office of R&D, NCSC,
Modeling and Verification and
Aids to Evaluation

Mr. H. Lubbes, Space and Naval
Warfare Systems Command

Mr. John Faust, Rome Air Development
Center (RADCOM)

Mr. John Preusse, Army Communica-
tions and Electronic Command
(CECOM)

THE STRATEGY

The DoD's CSP is aimed at a quantum increase in the security of America's automated information systems. Toward this, the National Computer Security Center (NCSC) has begun an aggressive, three-pronged R&D strategy. Its first major goal is to improve the security of current systems. Secondly,

the Center will encourage the development of new products using known technologies and finally will encourage new technology R&D.

The Computer Security RDT&E Program addresses the first priority by providing the means to test various security options or features, such as authentication, labeling, or auditing. For the second part of the strategy, the RDT&E program provides the technological support needed to achieve an "A1" class system, as defined in the DoD Trusted Computer Systems Evaluation Criteria. This includes stabilizing and improving verification environments, providing background material for refining security models used in the development of A1 systems, and finally, developing A1 demonstration systems themselves. The third phase of the strategy, i.e. going beyond A1 and transferring research breakthroughs into marketable products, depends entirely on the RDT&E Program.

RESOURCES

The Computer Security RDT&E Program is a cooperative undertaking led by the NCSC with the participation of the Army, Navy, Air Force, Defense Communications Agency, and Defense Intelligence Agency. Beginning with the FY84 budget, DoD RDT&E funds for computer security were consolidated, centralizing the program but permitting decentralized execution.

The FY88 program, as in the previous years, provides specific funds to be spent by the several DoD components. Consolidation, as prescribed in DoD Directive 5215.1 (the Computer Security Evaluation Center, October 25, 1982), avoids unnecessary duplication among DoD components; while decentralized execution takes advantage of the scarce expertise needed to provide technical oversight of contracts dealing with the highly technical field of computer security.

PROGRAM

To meet the challenge of transferring research breakthroughs into marketable products most effectively, we have channeled our efforts into five distinct areas: secure architectures, secure database management systems (DBMS's), network security, modeling and verification, and aids to evaluation. These five subprograms explore particular aspects of computer security research and development and, when combined, provide a solid program spiraling past the state of the art and into new technological frontiers.

Secure Architecture

Secure architecture addresses the design and implementation of trusted computing bases (TCB's). A TCB is the hardware and software

mechanism in a computer system that enforces security. Our current thrust is to push the edge of technology for TCB's by investigating kernel-based systems. Security kernels are the classical means of providing security in a TCB. They are a portion of the operating system that run in their own domain, separate from the normal operating system code, intercepting any operation that has security relevance.

The prolific growth of office automation and personal computer (PC) equipment and software within the Federal Government is another area of research concern. Little consideration has been given to the security aspects of these stand-alone and netted office automation systems. Nonsecure PC's, for example, negate the security provided by even the highest rated host because labels used within secure computers that indicate the security level of the data be lost once data is transferred to a PC. Security enhancement will be targeted at next-generation PC's since many of the current generation systems are single-state machines and cannot support security.

Security enhancement of existing commercial systems is a near-term solution. Under this task, we are incorporating security into the UNIX System V.

Advanced security architecture work provides new and different architectures for secure computers. The current effort in this area is the Logical Coprocessing Kernel (LOCK).

The LOCK takes a novel approach towards providing security in that it incorporates a separate security processor. (The LOCK effort is the subject of a separate paper of this conference.)

Placing the security mechanism in a separate processor has notable advantages over the kernel-based approach. The kernel is open to attack because its architecture shares security-related portions of the system with non-security-related parts. A separate security processor, however, prevents a user process from accessing the security-relevant portions of the system.

A by-product of security processors is improved performance, because they remove the security processing load from the main processor. The initial design phase of the computer should be available in 1988.

Secure Database Management

Multilevel database management security R&D has received far less attention than secure operating systems. In the summer of 1982, the Air Force and the National Science Foundation cohosted a workshop of experts in DBMS to examine the security problem. Three recommendations resulted:

*provide near-term relief, which is desperately needed and is achievable;

*for the mid-term, develop working demonstration of high-leverage applications; and

*conduct long-term research in the theoretical and practical foundations of secure multilevel DBMSs.

Although current and planned programs have made some progress towards achieving these goals, there has been no breakthrough that substantially improves DBMS security.

The Secure DBMS subprogram focuses on protecting databases and their related components. It is comprised of three research areas: trusted prototypes, studies and analyses, and advanced DBMS architectures. An effort to secure an existing DBMS entitled MISTRESS is now under way. Researchers are conducting various DBMS studies and analyses with the following objectives:

*data dependencies -- to achieve a family of multilevel secure DBMSs;

*evaluation -- to investigate the evaluation ramifications of DBMS's;

*sanitization -- to examine the downgrading and upgrading of multilevel data in database systems.

Another study is being conducted of the integrity lock technique. This cryptographically seals information stored in an automated system, with the objective of incorporating this technique directly into computer architectures supporting multilevel secure DBMS operations. Finally, the LOCK will be used to develop a trusted DBMS application.

Network Security

Network security focuses on the protection of data while it is being transmitted between host computers and users. A data communications environment has been created between geographically dispersed computers that includes networks of computers, terminals attached to computers that are attached to networks, and the internetting of multiple and various combinations of these systems.

Current computer networking technology has concentrated on providing services in a benign environment, and the security threats to these networks have been largely ignored. While literature abounds with examples of hackers wreaking havoc through access to public networks and the computers connected to them, hackers have exploited only a fraction of the vulnerabilities that exist. Techniques need to be developed that will prevent both passive exploitation (eavesdropping) and active exploitation (alteration of messages or message routing).

To reduce these vulnerabilities, we have initiated research in the development of components, high-level applications such as distributed processing, multilevel mail and file transfer, modeling, and advanced architectures. Within the area of advanced

architectures, we are conducting internet research, device authentication studies, and architectural simulation. The challenges facing us in the network security field are boundless. (The results of our network modeling work will be presented at another session of this conference.)

The NCSC hopes that coordination within the Federal Government and a sound R&D program will enable it to work with industry to create a line of network security systems that meet the needs of the Federal Government.

The problems of introducing computer security into the Ada programming language are being investigated. Ada is the DoD-mandated programming language for mission-critical systems. We are developing verification environments to be integrated into Ada software development systems as well as a suite of secure protocols in Ada to demonstrate how to marry these two technologies. (A special session devoted to these developments is a part of this conference.)

Evaluation Aids

Our Aids to Evaluation subprogram addresses the need to streamline and improve the system evaluation process. We believe we can make the evaluation process more responsive to our national demand for computer security requirements throughout the system's life cycle, identifying bottlenecks, automating tools to simplify the evaluation process, evaluating the effectiveness of safeguards, and reducing subjectivity in risk assessment. We are involved in research on intrusion detection evaluation tools and techniques, erasure and emergency destruction, risk management and generic product evaluation.

Modeling and Verification

Modeling and Verification explore conceptual solutions to computer security problems (modeling) and provide assurance that system specifications or implementations are consistent with the model (verification). R&D in modeling and verification addresses a critical need for trusted software and hardware systems of high reliability. To extend the state of the art in security modeling and verification approaches, we have embarked on five research endeavors: Ada verification, integrated design and verification environment, security modeling, software verification, and hardware and firmware verification. Our ultimate research goal is to verify systems at all levels of design and implementation.

CERTIFICATION: A RISKY BUSINESS

Martin Ferris
U.S. Department of the Treasury
Washington, DC 20220

Andrea Cerulli
National Security Agency
Fort George G. Meade, Maryland 20755

ABSTRACT

The Federal Government is the largest single producer, consumer, and disseminator of information in the United States[1]. Since government information is itself a commodity often with economic value in the marketplace, Federal departments and agencies are required to certify the protection of their automated information systems (AIS) that house sensitive information. Various government regulations and standards have only minimally described the certification process. Also, the manpower and money needed to make certification meaningful are scarce and reside primarily in special technical organizations.

This paper addresses certification in management terms, provides examples of certification in everyday life, and examines ways to maximize the use of national resources and policies to achieve a certified AIS application.

CERTIFICATION IN EVERYDAY LIFE

Life is full of risks. Most of us enjoy taking a risk every once in a while, whether that means a career change, a stock investment, or a bet on the Daily Double. Many risks, though, are transparent to us. To illustrate this transparency let's examine a typical weekday morning for most Americans. To get us through the day some of us will take vitamins, others valium, some both. A cup of coffee often is next for most "red-blooded" Americans. And half of the population dabbles in the fine art of make-up application before leaving the house. Now, we don't stop to think about whether the vitamins or valium are safe to swallow, the coffee grounds are pure, and the cosmetics are safe to apply. Instead, we disregard such thoughts because we entrust the quality of consumer goods to those whose responsibility it is to ensure the safety of such products. The mission of the U.S. Food and Drug Administration (FDA) is to enforce laws and regulations that protect the consumer's health, safety, and pocketbook[2]. The "Federal Food, Drug, and Cosmetic Act" is the basic food and drug law of the United States to assure the consumer that food is wholesome and safe to eat; that drugs are safe and effective for their intended use; and that cosmetics are safe and made from appropriate ingredients. We trust the FDA to certify that products are safe. For instance, coffee imported into the United States is inspected for infestation, mold, and contamination, and if found objectional, the cargo is refused entry. Vitamins, valium, and cosmetics are also protected under the Act against misbranding or adulteration. Because the FDA enforces rigorous regulations to protect consumers, the FDA's approval of a product assures us of its safety.

Like the FDA, the Public Health Department enforces regulations to ensure a clean and healthy environment in public places. Before facing the work day, some of us might stop at a diner or fast food restaurant for breakfast. Most restaurants today maintain high standards of cleanliness not only due to Health Department regulations, but also because the public will not tolerate dirty, insect-infested eateries. The procedures to maintain a healthy, pleasant atmosphere remain transparent to the customer since much of the maintenance is performed after-hours or behind-the-scenes. The restaurant owner relies on the Health Department's certification to assure the public of a healthy, safe environment and also to continue business.

Although we will probably not jeopardize our lives by drinking coffee, wearing lipstick, or eating an "Egg MacMuffin," we do risk adverse reactions if any of these products are not inspected or prepared properly. They are all vulnerable to either accidental or malicious tampering, whether performed by people, insects, or machines. Automated information systems (AIS) are also vulnerable to accidental or malicious tampering which could cause unsafe operations. The vulnerabilities could present unacceptable risks to computer applications which require protection of its sensitive information. Just like vulnerabilities in consumer goods, AIS vulnerabilities must also be managed.

CERTIFICATION OF AN AIS

Now the AIS is becoming a part of

everyday life and of course bringing along with it associated vulnerabilities and risks. Managers of AIS resources must be prepared to face risks of disclosure, modification, and non-availability of their information. Files that were once kept private within the confines of a physical office space are now vulnerable to uncontrolled access. Managers responsible for AIS resources need confidence that reasonable assurances (or acceptable levels of risk) are applied to AIS resources. And, if and when a vulnerability is exploited maliciously or accidentally, the manager wants to turn to someone who can explain why it happened. Just as in everyday life, managers need the best information available to establish confidence and accountability in business operations.

For federal AIS managers, providing reasonable assurances for the protection of AIS resources is essential to assuring the integrity of federal operations. Federal managers are required by law, the Federal Managers' Financial Integrity Act[3], to provide reasonable assurances for federal resources. Confidence and accountability are required for the protection of federal resources from fraud, waste, and abuse. Not arbitrary, this policy is looking out for the true owners of federal resources - the public. The Office of Management and Budget (OMB) Circular A-123, "Internal Control Systems[4]," implements this law. A-123 is an internal controls regulation that sets in motion a process to establish confidence and accountability in the protection of federal operations from fraud, waste, and abuse. To achieve this process, A-123 requires management control plans based upon such actions as vulnerability assessments, personnel performance agreements, and annual letters to Congress stating whether reasonable assurances are being applied.

OMB Circular A-130, "Management of Federal Information Resources[5]," is a separate regulation that establishes requirements for the effective and efficient management of federal information resources. This regulation is the Executive Branch's response to several information-related laws including the Paperwork Reduction Act, the Privacy Act, and the Freedom of Information Act.

A-130 also requires that all agency information systems possess a level of security commensurate with the sensitivity of the information and also commensurate with the risk and harm that could result from improper operation. Furthermore, the manager whose program an information system supports is responsible and accountable for the products of that system.

More specifically, Appendix III of A-130 requires that Federal agencies establish AIS security programs to safeguard sensitive information processed by their AIS. This appendix requires an AIS security program to consist of four parts: applications security, personnel security, information technology security, and a security training and awareness program. As part of applications security, an appropriate "agency official" shall certify all sensitive AIS safeguards

based upon the appropriateness determined by risk analysis, a design review, and system testing. Periodic reviews are required to preserve the integrity of previous AIS certification decisions. The periodic reviews not only serve as a security requirement, but also as a necessary way of preserving the investment of previous certification decisions.

The relationship among the OMB Circulars is sometimes overlapping. Assuring effective and efficient information resources management is a requirement that supports the reduction of waste and abuse. However, pursuing effective and efficient AIS operations can produce high risk to the AIS resources. For instance, organizing all files into a common AIS data base may be more effective and efficient, but the risks to privacy, fraud, and abuse may be significantly higher. Consequently, continuous coordination is required among those responsible for implementing the Circulars.

Fundamental to both effective and efficient operations and secure operations is the security program outlined in Appendix III of A-130. Appendix III serves as a security requirements tool for the rest of A-130 and A-123. It also serves the internal control needs of OMB Circular A-127, "Financial Management Systems[5]," which establishes a program to assure the integrity of federal financial management systems. Consequently, a credible certification statement is fundamental to responding to federal regulations. Certifying the confidence and accountability of the protection provided to AIS resources is a basis for many management-approval processes. Certification is also a fundamental tool for the managing of federal operations where sensitive information is processed by an AIS. This management view demonstrates the growing dependency of implementing various internal control laws and regulations on AIS security certification. It also shows that a meaningful certification decision requires coordination between management and technical communities within a Federal agency.

Viewing AIS certification as a fundamental management tool also shows how important the completeness and integrity of technical information supporting certification decisions must be. Gathering complete and consistent information for certification decisions is difficult work and often requires the services of technical specialists. Gathering both technical and business-oriented information involves much analysis to identify, understand, and control the vulnerabilities and threats to the AIS and its application(s) in question. With the mandate to federal information resources managers to make information systems more efficient, reliance on complex technical AIS controls make simple AIS certification decisions unlikely. A fully documented and informed certification decision would include analyzing and controlling the AIS

from applications software and operating systems, to microcode and hardware throughout the AIS development and life cycle. Lesser documented and informed certification decisions increase the risk of insecurity while at the same time threatening the investment made to reach the certification decision.

The ongoing system development and life cycle of the AIS and AIS applications provides the best opportunity to acquire the best AIS technical information. This information is needed for consideration of technical decisions whether they are security- or operational-related. Ideally, the AIS should be built to specific security requirements. Independently developed security features added on to an AIS present the potential for additional vulnerabilities and risks if such features are not consistent with the objectives of the system being augmented (see reference 6). The relationship between the AIS and add-on features is easier to understand and control when adding some features such as cryptographic processes to the AIS communication subsystem. Other times it is harder to understand and to assure control as when adding an access control package to a particular operating system. Since most federal managers have no control over the development and life cycle of commercial AIS resources, they can only accept what the commercial market can provide, often times adding on security features.

Specifying appropriate security safeguards, assuring that they are properly designed, assuring that their implementations are adequately tested to meet their design, and assuring that they make sense in the context of the entire AIS is a critical process. If qualified and experienced personnel are not involved with such undertakings and a comprehensive approach to safeguards is not taken, then the resources spent on acquiring the safeguards may have been wasted.

Given all the technical and management complexities in acquiring the best information for a certification judgment, there should be a number of thoughts that run through a senior information resources manager's mind when planning a certification program for his or her AIS resources. Some thoughts to consider if you are a manager follow:

How much resources should I commit to this problem? A National Bureau of Standards (NBS) Publication, Overview of Computer Security Certification and Accreditation, notes that full organizational commitment must exist for the training and support of a security program to perform credible certification analysis. Unfortunately, budgeting for an activity that doesn't show a tangible return is hard to justify. After all, even if it were certified based upon the best information, that wouldn't necessarily prevent something "bad" from happening. And even if you do budget for a certification analysis, there is no guarantee that it will survive the various levels of a federal budget process.

How many technical security experts and AIS resources does my organization need to implement a meaningful AIS security program based upon credible certification decisions?

How much continuous security education and what types of security education do my personnel need to make my AIS security program credible? Do I have to support a crew of cryptographers and secure operating system engineers?

How often must I exert the necessary resources to make my AIS security program credible?

Does my organization have the resources to repeat a certification process for 600 computers scattered across the country that have similar but different applications?

And, if I can't make my AIS security program credible do I stop pushing the efficiency of my AIS resources? Or do I push forward and accept more risk?

The federal AIS manager who must think about these questions must also have the best information available to make intelligent decisions. This information is hard and expensive to get without help. In fact, it may be impossible to get ... without help.

So, an AIS manager's level of maturity is tested. The renowned behaviorist David McClelland claims in his book, Power: the Inner Experience, that managers are motivated by the need to influence and that a mature manager can apply influence in both personal and social ways[7] with social influences being the most effective. This requires the AIS manager to articulate common objectives and recognize her or his limitations in fulfilling those common objectives.

This means becoming a team player. AIS managers must look for teammates that complement each other in pursuit of group objectives. Together the teammates will provide the means to an end. How does a AIS manager find her or his teammates?

First, knowing your strengths and weaknesses is a start. By dividing AIS security problems into manageable portions, an AIS can be viewed as a collection of various components, some of which may be overlapping. The AIS components could include technical components (computers, terminals, modems, communications systems) as well as non-technical components (operator, security manager, procedures, applications, and user). The components may be described as application types of components (parts ordering systems, financial systems, law enforcement systems, data base systems). Also, an AIS security program can be divided into various components. Appendix III of A-130 states certification consists of risk analysis, security specifications, design reviews and testing. When considering the various AIS and AIS security components, a manager can

view what can be affordably and directly controlled under a AIS security program and what can not.

Expressing these components in a common language is another important step. It is becoming increasingly important for AIS managers to be familiar and active with the various standards communities, especially the industrial standards community since they potentially have the greatest commercial effect. Even though standards tend to allow for some broad interpretations, often to accommodate existing investments, and even though they take a long time to mature, they nevertheless provide a means of communicating applications as well as technical details in a common language. Various American National Standards Institute (ANSI) and Institute of Electrical and Electronics Engineers (IEEE) standards are helping AIS managers describe the technical functioning of various AIS components in a language that promotes consistency of those functions. Various standards, including ANSI X9 financial service standards and ANSI X12 business data interchange standards, are providing a common way to describe or specify security features as a part of everyday business transactions.

By surveying the standards market the manager can tell which standards apply to his or her components. Consideration of a standard's relevance to an AIS application(s) is important. So is consideration of the means to validate compliance with the standard. Although validation of security standards is not enough to claim that the standards implementation is secure, it is a useful step in screening implementations that claim to meet the standards that may not.

Standards and standards validation are different from evaluations. Standards are broad requirement statements of security features, sometimes with enough options to make validation feasible only if a subset of options are validated. Evaluations are more implementation specific. The Federal Communications Commission provides evaluation support for some communications components. Underwriters Laboratories provides evaluation support for some non-technical components such as fire extinguishers. The manager should consider the national evaluation programs as the team player who assures implementation details are consistent.

HOW NATIONAL RESOURCES CAN HELP THE MANAGER

Just as managers should use standards to help define their security requirements, they should also make use of the national resources which evaluate various subsystems' compliance with particular standards. There are three national resources addressed below; two of the resources evaluate and endorse or certify subsystems, while the other only evaluates subsystems. The difference between evaluated and endorsed/certified products is subtle to the manager certifying the applications. AIS subsystems that are evaluated, but not certified place more accountability on the user of the product and the management of the product's vendor.

The manager should recognize that the certification of her or his AIS application may rely upon similar but isolated certifications of the AIS' subsystems or shared systems. For example, the AIS may include an operating system, a data base system, an access control system, an encryption or authentication system, an entire computer system or networks. These national-level programs will evaluate or certify various subsystems that the manager might want to consider as part of his or her AIS application. But these various subsystems supporting an AIS application must have common security objectives. Consequently, managers must ensure that common security requirements and standards are required for each subsystem. The evaluation, endorsement, and certification programs described below are available to the manager and are sponsored by the National Computer Security Center (NCSC), the National Security Agency (NSA), and the U.S. Department of the Treasury.

Under the Commercial Products Evaluation Program, the NCSC performs computer security software and hardware product evaluations on commercial security products. The NCSC does not certify these products, but does place those products that meet evaluation requirements on the NCSC's "Evaluated Products List (EPL)." Managers can "shop" off the EPL and be assured that these products have been extensively evaluated. The standard the NCSC uses is the Trusted Computer System Evaluation Criteria (also known as and referred to hereon as the "Orange Book") [8]. Vendors can opt to have their products evaluated at different levels of security such as discretionary access protection, controlled access protection, mandatory protection, structured protection, and verified protection. Each level of security guarantees certain protection features. For example, if the NCSC evaluates an access control system and it meets the Orange Book requirements at the level of controlled access protection, the manager can be assured that the system will include the following features: audit trail capabilities, object reuse, user identification and authentication, and discretionary access control. The types of subsystems that the NCSC has evaluated so far under this program include operating systems and add-on packages such as access control systems. The NCSC estimates that to evaluate a product at the controlled access protection level of security requires four people working a quarter of their time for one year or one-man-year. The man years increase as does the level of security in the product. Two man-years is the estimated time required for structured protection, which includes all security features in the lower levels and also mandatory access control, labeling, and the reference monitor concept in the operating system.

For those vendors who would rather not commit to a full-scale evaluation, the NCSC sponsors a Subsystem Evaluation Program in which the vendor selects the security features it wants evaluated. For example,

a vendor can request a product evaluation of access control or object reuse capabilities instead of committing to a full-scale evaluation. Products that have been evaluated under the Subsystem Evaluation Program are special purpose products such as user identification and authentication devices. Products from both of these evaluation programs provide a cost-effective way for managers to choose their subsystems according to their security needs. Another program that not only evaluates but also endorses devices is sponsored by the NSA.

NSA sponsors the Data Encryption Standard (DES) Endorsement Program, also known as the Federal Standard 1027 Program. Over the past ten years NSA has endorsed over 35 DES products for both voice and data applications. Managers who require such devices can choose from a variety of manufacturers to suit their needs. NSA has decided to phase out the DES Endorsement Program and will no longer accept new products for evaluation and certification after January 1988. In replacement of the DES Endorsement Program and also to foster new business relationships with the U.S. telecommunications industry, NSA began its Commercial COMSEC Endorsement Program (CCEP) in 1985. The objective of the CCEP is to provide a widespread availability of quality, inexpensive, secure telecommunications systems for use by both the U.S. Government and the private sector. The products developed through the program will employ NSA-proprietary, classified cryptography. The implementation of this cryptography, however, will result in unclassified products. Vendors can design products for use to secure classified information or for use to secure unclassified only information. So far, various large and small corporations have signed 37 contracts with NSA to design secure products. Four products have undergone endorsement, which is the final phase of the CCEP before production.

NSA has given one exception pertaining to the DES Endorsement Program; NSA will continue to support DES devices for financial applications under the U.S. Department of the Treasury's Electronic Funds Transfer (EFT) Certification Program for Authentication Devices. NSA stated in a memorandum to Treasury, "We agree ... with continued Treasury certification of DES equipment until transition to new cryptographic technology is possible." NSA also stated it will continue to support Treasury's program with technical guidance and assistance. In addition to the technical expertise NSA provides to the program, the National Bureau of Standards (NBS) also plays a role. A more detailed description of how Treasury's program can benefit the manager follows.

TREASURY INITIATIVES TO IMPROVE THE CERTIFICATION PROCESS

The U.S. Department of the Treasury has revised its AIS security program to make AIS security more affordable, simple and meaningful for all of Treasury's twelve bureaus. Treasury has made certification the end that risk analysis and security

specification's design reviews and system testing serve. At the same time it has begun a management study to determine how AIS certifications can best serve the other management control processes. Treasury's strategy for improving the AIS security program is to use existing industry standards and evaluation programs to maximize the cost-benefits to Treasury's AIS security program, while acquiring cost-effective AIS security. Treasury's continuing involvement and commitment to the ANSI and federal standards processes represents an investment in the future development of standards that satisfy Treasury's operational and security AIS needs.

Treasury's initiatives begin with departmental policy. Several policy decisions have partitioned the AIS security program resource problem into manageable parts. The first policy is Treasury Directive 85-01, entitled, "Information Systems Security[9]," which simply defines three categories of information-classified, sensitive unclassified, and public - that can be processed by a Treasury AIS. This break-out provides a high-level framework to determine minimum levels and types of safeguards needed for each category of Treasury information.

The second Treasury Directive, TD 85-02[10], deals specifically with sensitive unclassified AIS information. TD 85-02 defines Treasury's Automated Information System Security and Risk Management Program as required by OMB A-130, Appendix III. This directive establishes acceptable risk for the department in terms of the implementation of minimum security requirements. The policy and its associated handbook is a product based upon the coordinated input of all twelve Treasury bureaus and the advice and guidance of the NCSC. The bureaus will base their AIS security programs on these minimums. Because of the varying sensitivity of AIS resources and the availability of AIS security program resources, some of Treasury's bureaus will choose to do more than the minimum. Meanwhile, the baselines provide a focus for the AIS security program; a basis for AIS security education, risk analysis, security specifications, design reviews, and security testing.

The minimum security requirements are based upon existing standards. The standards chosen include controlled access protection (the "C2" level of security as defined in the Orange Book) for computer security; and NSA-approved cryptography for data communications, whether DES-based or CCEP-based cryptographic products. Besides making technical security sense, these standards were chosen because they also provide a management tool to reduce the AIS security program costs to obtain a meaningful certification. This is due to the fact that, as mentioned earlier, NSA and NCSC experts have evaluated the AIS subsystems by reviewing the design and testing the implementations of the respective standards. Moreover from an AIS

security program perspective, these evaluations make up a continuing program of configuration control assuring that participants in the evaluation programs maintain the security for the life-cycle of the product. Although there are never any guarantees that careful design and evaluations will fully remove the vulnerabilities or that the commercial participants will fully comply with the NSA or NCSC programs, if there is a problem, there are national resources and national programs to help.

A third policy position applies to a specific AIS application - Electronic Funds Transfers (EFT). In 1984 Treasury began developing a policy on EFT Security. It was determined that the best existing countermeasure was authentication in lieu of encryption due to the international scope of the requirement. ANSI X9.9, "Financial Institution Message Authentication," was selected as the standard. Treasury's EFT policy, entitled, "Electronic Funds and Securities Transfer Policy -- Message Authentication and Enhanced Security[11]," requires that all Federal Government EFT transactions be protected using message authentication by June 1988.

The Department established an EFT Task Force composed of representatives from diverse government agencies to develop criteria for certification of authentication devices. The criteria is based upon industry standards including ANSI X9.9, ANSI X9.17 on key management, Federal Standard 1027 (DES), and ANSI/IEEE 829 Standard for Software Test Documentation. The criteria were published in May 1985 and have been sent to over 250 interested parties and corporations. Since then, through Treasury's EFT Certification Program for Authentication Devices, the Department has been working with various vendors to guide them through the development of devices to meet the certification criteria.

Treasury certified its first device in June 1986 and is currently working with several more vendors who are developing authentication devices. The Department is in the process of expediting implementation of EFT authentication on its financial systems. Treasury received national resource assistance in this certification program by signing a Memorandum of Understanding with NSA and NBS. NBS agreed to provide support in validating compliance with various security-related standards. This has included ANSI X9.9, ANSI X9.17, the Data Encryption Standard, and software engineering standards. Of course, NSA security evaluation support is mandatory because no one else in the Government has their expertise.

Treasury has reimbursed NBS for developing automated validation systems to validate vendor compliance with ANSI X9.9 and ANSI X9.17. At this time, NBS has completed the ANSI X9.9 automated validation systems for Treasury and is expected to complete portions of the ANSI X9.17 automated validation systems by the end of the year. Thus far, eight vendors have completed the

automated validation of their products' compliance with ANSI X9.9, one vendor product has been certified for Federal use, and three others have entered Treasury's program for certification.

So, Treasury will make certification decisions on EFT authentication devices based upon the best information available from the results of standards evaluations and implementation evaluations. These certified products will be a basis for certifying the EFT applications implemented on federal AIS. Federal managers of EFT functions will have a tool that will reduce their security program expenses of complying with A-130, Appendix III. Moving up the federal internal controls ladder, these certified AIS applications will provide Federal agencies a basis for assurance that their financial systems comply with the objectives of A-123 and A-127 as well as will provide the basis for more effective and efficient processing of financial information by removing much of the current paper flow. Of course, this doesn't totally remove federal agency AIS internal controls responsibilities where EFT is processed. Proper administrative internal controls such as separation of duties must be factored into those systems.

If other AIS applications exist on those systems that have sensitive information to be processed, a certified AIS subsystem exists on the system that can be used to assure data integrity and possibly confidentiality as well as user accountability. The Consolidated Data Network (CDN) is Treasury's effort to provide effective AIS services to its bureaus throughout the country. The CDN will be a totally encrypted DES network, which will make it the largest encrypted data system in the civil Federal Government. It will grow to be the Department's secure data communications utility.

The network is currently being link-encrypted using NSA-endorsed DES devices. As end-to-end types of protection become available, they will reduce much of the security product needs. As enhanced key management technologies become available and are implemented, whether ANSI X9.17 or other techniques, Treasury's security and operational costs will improve.

But, again, there is good news for the various AIS applications such as tax processing, revenue collection, law enforcement, payroll, personnel and many other Treasury AIS applications. They have another certified AIS component at their service and another tool to base AIS certification decisions on without spending a lot of resources designing, and testing cryptographic safeguards. Although CDN will not provide all the protection that some users might need in the near term, a focus for their AIS planning to address those other security needs is provided for them.

CONCLUSION

To make an intelligent certification decision the AIS manager needs the best information available. That involves gathering both technical and business-oriented information to make a cost-effective decision. The resources for this information are scarce or out of the direct control of most AIS managers. On the surface, the problem appears ridiculously difficult. But, a smart manager will see the problem as similar to everyday life and should try to create a similar environment. Managers must know their strengths and weaknesses; what they can directly control and what they have to rely upon others for help. They must also become a member of a larger standards community. All managers should be strong in knowing what their information resource requirements are. They can help the community by expressing these requirements in the form of community standards. For significant portions of their AIS security requirements, services offered by national resources can help the manager in areas of technical expertise.

REFERENCES

1. OMB Circular No. A-130, "Management of Federal Information Resources," December 12, 1985.
2. U.S. Department of Health and Human Services, Public Health Service, Food and Drug Administration, Requirements of Laws and Regulations Enforced by the U.S. Food and Drug Administration, Washington, D.C.: Government Printing Office, 1985.
3. Federal Managers' Financial Integrity Act of 1982, Public Law 97-255, September 8, 1982, 97th Congress.
4. OMB Circular No. A-123 (Revised), "Internal Control Systems," August 4, 1986.
5. OMB Circular A-127, "Financial Management Systems," December 19, 1984.
6. U.S. Department of Commerce, National Bureau of Standards, Overview of Computer Security Certification and Accreditation, NBS Special Publication 500-109, Washington, D.C.: Government Printing Office, 1984.
7. David C. McClelland, Power: the Inner Experience, New York: Irvington Publishers, 1975.
8. U.S. Department of Defense, Department of Defense Trusted Computer System Evaluation Criteria, December 1985.
9. U.S. Department of the Treasury Directive 85-01, "Information Systems Security," April 2, 1985.
10. U.S. Department of the Treasury Directive 85-02, "Automated Information Systems Security and Risk Management program," April 24, 1987.
11. U.S. Department of the Treasury Directive 16-02, "Electronic Funds and Securities

SECURITY EVALUATIONS OF COMPUTER SYSTEMS

David J. Lanenga
National Computer Security Center
9800 Savage Rd.
Ft. George Meade, MD 20755-6000

INTRODUCTION

This paper describes the process of computer security evaluations as presently performed by the National Computer Security Center (the Center). This subject is important for a number of reasons. The first is that, because the Center has organized the evaluation process, there are many others who may benefit from sharing this information. There are many organizations that evaluate or certify system security, or that are involved in planning for a certification. What the Center's evaluators do is not significantly different from what these groups do, and the process used by the Center's evaluators is something that can be adapted for use by others. The Center has organized the process so that it can be controlled and managed. This paper describes how this was accomplished, what the Management Plan consists of, and some of the details of the evaluation process.

A second reason for this subject is that so many vendors and developers have asked questions such as, "What do you do in an evaluation?" and "What does an evaluation of a computer product consist of?" I hope to answer those kinds of questions and hope to provide an understanding of what happens during an evaluation.

INITIAL PROCESS

In the beginning, computer security was something of a void. The Center's purpose was to provide a list of evaluated products that the Federal agencies could purchase off-the-shelf, with the knowledge that the product met a certain standard of security.

The Center was formed in mid-1981, and the first secure product evaluations began late in 1982. Evaluations really picked up in the following year when the Criteria was published. At that time, the evaluation staff consisted of only five evaluators from the Center, augmented by additional evaluators from the MITRE Corporation and the Aerospace Corporation.

At that time, the evaluation process didn't really exist, because nobody had ever tried to do an evaluation like this before. It was a totally new procedure. The evaluators didn't even have a final version of the Criteria at the start. In addition, a strong concern for quality hampered the development of evaluation procedures. Center management closely reviewed the evaluation work and draft evaluation reports to ensure the level of quality in an evaluation because they felt that acceptance of the Center by the

computer industry depended heavily on the first evaluation efforts.

THE MANAGEMENT PLAN

During the first evaluations, it appeared that the evaluators weren't doing evaluations very efficiently, and that it took too long to complete an evaluation. Because no planning and management tools were in place, it was not possible to measure efficiency, effective use of resources, or adherence to schedules.

In order to improve the evaluation effort, the Center's Evaluation Division has sponsored seven semi-annual Evaluators' Workshops since September 1983. The Workshops are held to discuss interpretations of the Criteria, to share experiences in evaluations, and to resolve evaluation issues faced by the evaluators.

In October 1984, following one of these workshops, the Aerospace Corporation was commissioned to develop a Management Plan for the evaluation effort. This document was developed through an analysis of past evaluations. Every evaluator associated with the Center participated in developing the plan, and the plan was issued in October 1985. It included all the things that had been done correctly, omitted the things done incorrectly, and was general enough to leave room for all the things the evaluators have learned since then. The Management Plan turned out to be a real success. It made an evaluation a much more orderly process.

One purpose in developing the Management Plan was to help the evaluators PLAN evaluations, something that had not been done very well at all. This was to be expected. The evaluators were skilled in areas such as security, computer technology, and operating systems. Most of them had very little exposure to formal planning and didn't really want to know any more about it either. The Management Plan solved this problem. It detailed all the tasks that are a part of an evaluation. It also provided a list of tools and reference material available for each task, and enumerated factors which should be taken into consideration when calculating the duration of each task. The Management Plan is not merely a checklist. It is a resource used by the evaluators to help them decide which tasks are appropriate, how they relate to each other, in what order, and how long they can be expected to take.

Another purpose of the Management Plan is to provide a measure of control to the

process. The earlier evaluations were driven by the system developer's schedule, which obviously includes many considerations other than security. The developer is in business to make a profit, and must use his resources efficiently in order to do so. Without a plan, the evaluation process was geared to the vendor's schedule, and the Center had no control over the schedule. Because the Center is spending taxpayers' money, it must also use its resources efficiently. By presenting the system developer with a plan for an evaluation, and by showing that it is a reasonable plan, it is possible to prepare a mutually agreeable schedule and adhere to it. As a result, everybody involved in the evaluation process - including Center management, the vendor, and the evaluator - is happier.

When things somehow fail to go according to the plan, as they seem to do in any endeavor, tools provided under the Management Plan alert Center management to assist the team and the vendor to get things back on track. The Center managers need and use inputs from both the team and the vendor. Just as the team may have problems with the vendor's ability to meet their needs, the vendor may disagree with the team's interpretation of the Criteria, the speed at which it appears to be working, or perhaps its ability to understand the vendor's point of view. The Management Plan has built-in feedback loops through successively higher management levels to resolve team/vendor issues and bring the evaluation to a successful end.

EVALUATION PHASES

In organizing the evaluation effort, the Center first divided the evaluation effort into two distinct parts. The two parts have been known as the Developmental Phase and the Formal Phase of an evaluation. The first part is currently called a design analysis, and the second is called an implementation analysis.

The two phases of an evaluation are substantially different. The Center wants to be involved with a developer at the beginning of the design stages of a new system, when there is the greatest opportunity to influence the design, or at least the security aspects of the design. It's to the advantage of the vendor, too, because correcting design flaws is increasingly more expensive as one proceeds with the design process. The problem at this stage is that there is usually little or nothing to evaluate. It's difficult to do a rigorous, technical evaluation of something that doesn't yet exist. The second part of an evaluation, the implementation analysis phase, is something that should be completed as expeditiously as possible. This is to the advantage of both the Center and the vendor. When the vendor HAS a final product, and is fully prepared to provide ALL the evidence necessary to show that it is a secure product, then the evaluators want to examine that evidence to the best of their

ability and bestow a rating as quickly as possible.

DESIGN ANALYSIS PHASE

The design analysis phase of an evaluation is a consulting relationship. The members of a design analysis team are the most experienced evaluators. They are able to assess the consistency of the design against the requirements of the Criteria. The design gives the team a solid assurance as to how well the requirements will be satisfied. The team members can answer the questions such as "Is this good enough?" or "On a scale of 1 to 10, where do I stand?"

The central task in a design analysis is called Systems Analysis and Technical Support, and is performed through technical interchange meetings with the vendor. The level and nature of support vary widely. Critical factors in determining the level of support are the vendor's experience, candidate level of the Criteria, and whether the product is an existing product or a totally new design.

The Management Plan provides very detailed guidance to both the evaluator and to management, while still allowing for judgement by the team leader. A small sample of tasks that make up the design analysis phase of an evaluation are the following:

- o Develop Verification Plan (B2 +)
- o Develop Training Plan (for formal team)
- o Determine Configuration Range
- o Analyze Documentation (draft ok) as it is developed
- o Educate the vendor's technical staff
- o Determine when ready to prepare the Initial Product Assessment Report (IPAR)
- o Determine Candidate Evaluation Class

Each task in the Management Plan is subdivided into a set of subtasks whenever possible. For example, the task labelled analysis of system documentation is subdivided by the individual requirements for documentation:

- o Formal Top Level Specifications (FTLS)
- o Descriptive Top Level Specifications (DTLS)
- o Formal Security Policy Model (e.g. Bell/LaPadula)
- o Security Features User's Guide (SFUG)
- o Trusted Facility Manual (TFM)
- o Covert Channel Analysis
- o Test Plan

In order to document the first phase of the evaluation, the evaluation team writes an Initial Product Assessment Report (IPAR). This document assures that all Criteria requirements have been addressed,

and that the IPAR contains sufficient product information to form a basis for the decision regarding whether to proceed to a Formal Evaluation, or Implementation Analysis. It documents the justification for the candidate rating, the team's understanding of the product, and the vendor's understanding of the Criteria.

This completes the first phase of an evaluation - the design analysis phase. Portions of the evaluation dealing with administration and management review of the process have been omitted in order to focus on the technical areas, but they are definitely a part of the Management Plan.

IMPLEMENTATION ANALYSIS

The second phase of an evaluation is called the implementation analysis phase. It was formerly known as the formal evaluation, and it is what most people think of when they think of an evaluation.

The description that appears below is common to all the evaluation work performed by the Center and applies equally to sub-systems evaluations, operating system evaluations, network evaluations, and evaluations of database management systems. The guiding principle is that the vendor provides all the evidence needed to judge the quality of security in a given product, and the evaluation team analyzes that evidence.

The distinct elements of the implementation analysis are:

- o Planning
- o Education
- o Analysis
- o Draft Final Report
- o Test Plan
- o Testing
- o Configuration Management Review
- o Final Report and Rating

Planning: The very first thing that happens in an evaluation is the development of a work plan for the evaluation. This plan is developed by the evaluation team, agreed to by the vendor, and approved by Center management. Adherence to the plan is enforced through regularly scheduled reviews by Center management. This is done through informal reviews and briefings, meeting reports, trip reports, and monthly status reports submitted by both the team and the vendor.

Education: It's important that the evaluation team fully understand the functionality and interfaces of the product to be examined. Although the original team helped the vendor define and schedule the training plan, the training doesn't occur until the second phase of the evaluation. Education is important in this phase because the analysis occurs on a much greater level of detail.

Analysis: The team analyzes the documentation that was reviewed during the first phase of the evaluation, but at a

much greater depth. The documentation analysis is followed by an analysis of source code because, after all, this IS an implementation analysis. The team must be certain that the design has been implemented, and implemented correctly. They need assurance that the system actually works as advertised. The Management Plan is still incomplete in this area, and there is an on-going effort to define this process more rigorously and in greater detail. In general, individual team members pursue areas of documentation and code that correspond to the various sections of the Criteria. The approach varies, depending on the target rating. At the lower levels of the Criteria, evaluators are primarily concerned with functional mechanisms, such as discretionary access controls, auditing, and identification and authentication. At the higher levels, the assurances provided through system architecture, configuration management, and formal verification are more important.

Draft Final Report: Throughout the previous steps, beginning in the education phase, team members take notes and mentally organize the sections of the final report for which they are responsible. As their understanding of the system grows, the first draft of the final report is written.

Test Plan: The test plan is the work plan for the system testing phase of the evaluation. It describes the functional tests to be conducted and specifies test procedures. As in all the sections of the Management Plan, this section incorporates the flexibility to deal with evaluations at any candidate class of the Criteria. For example, for B2 level systems and above, the test plan includes the penetration testing methodology and any testing related to the vendor's covert channel analysis. The plan provides a schedule for testing, identifies the test site, and describes the system configuration to be tested.

Testing: In order to support the assurance obtained through analysis of documentation and code, a certain amount of testing must be done. The objective is to execute security-related functional tests for the candidate system. The team examines the vendor's functional tests and evaluates the results. Where necessary, the team develops additional tests to ensure that all of the features are adequately tested. When errors are found, the vendor is expected to correct them. The team documents its findings in the final report.

Configuration Management Review: The Center's purpose in configuration management is to ensure that changes to the system can be traced from beginning to end, and vice versa. The evaluators should be able to trace a trouble report all the way down to the exact location of code changes. They should also be able to trace code changes back to the reasons for the changes. If it is known what changes have taken place, their effect on the security of a system can be assessed. Although the Criteria doesn't require configuration

management until the B2 level, the Center now looks for it on all systems as a practical matter. The Center is very reluctant to consider maintenance of an initial rating over subsequent releases of a product unless an approved configuration management system has been implemented.

Final Report and Rating: When the testing has been completed, the team knows the product as well as it is ever going to, and is ready to complete the evaluation. The draft final report is modified based on the recommendations of the Technical Review Board (TRB) and the additional information learned during system testing. The evaluation is complete when the Center's rating is awarded, the product is entered on the Evaluated Products List, and the Final Report is published.

Throughout the course of an evaluation the team has ready access to technical specialists within the Center. These people are the Chief Evaluator, Senior Scientist, and Chief Scientist. These people are involved daily with the twenty-five evaluations currently in progress. In addition to providing technical expertise, they are also able to help the team separate technical issues from administrative and management issues. All this helps to keep an evaluation on course.

The quality control function mandated by the Management Plan is the Technical Review Board, or TRB. The primary purpose of this board is to verify the team's depth of understanding of the product under evaluation and to assure consistency with other evaluations. The TRB may approve the team's progress, or it may recommend that

the team investigate some areas more thoroughly before proceeding to the next phase.

The TRB reviews the work of the evaluation team at least three times during an evaluation. One review takes place at the end of the first phase of an evaluation, when the team presents its Initial Product Assessment Report. The second is a review of a draft of the final report and the team's test plan. The third review is at the end of an evaluation, when the final report is reviewed. At each of these reviews, the team provides the TRB with a document that represents a great deal of work. The TRB members review the information provided, provide comments, and ask questions about the conclusions the team has formed. The team responds to these comments and questions in a formal presentation. The TRB judges the presentation of the team in the light of previous evaluations, and makes recommendations to the Chief of the Product Evaluations and Technical Guidelines Office, who makes final decisions regarding the future of an evaluation and the course to be taken by the evaluation team.

Conclusion

Through its development and implementation of the Management Plan, the Center has demonstrated that evaluation activities can be planned, scheduled, and managed. The Center's activities closely mirror normal contractual and especially certification/accreditation activities. With minor changes to particularize this plan to other organizations, it can serve a wide variety of similar functions.

AN EXPERT SYSTEM APPROACH TO SECURITY INSPECTION OF A VAX/VMS SYSTEM IN A NETWORK ENVIRONMENT

Henry S. Teng and Dr. David C. Brown
Digital Equipment Corporation Artificial Intelligence Research Group
77 Reed Road, HL02-3/C13 Computer Science Department
Hudson, MA 01749-2895 Worcester Polytechnic Institute
Worcester, MA 01609

ABSTRACT

We have developed a prototype expert system, named XSAFE, for computer security inspection of a VAX/VMS system in a network environment. XSAFE attempts to explore the vulnerabilities of a given VAX/VMS system using a remote diagnosis mechanism. The inspection results provide valuable information to system management about further security improvements.

The computer security inspection is performed by four security inspectors: the Password Inspector, the DECnet Default Account Inspector, the System File Protection Inspector and the User Application Inspector.

User application security is the focus of the development of XSAFE, since it is the weakest component of a VAX/VMS system from a security point of view.

XSAFE has been field-tested on Digital's internal network and has produced some very encouraging results. The field test results have clearly shown the potential of XSAFE as a centralized security auditing system in a distributed network environment.

1 INTRODUCTION

XSAFE* is a prototype expert system that can assist a system manager or network manager in the inspection of the security of a VAX/VMS** system in a network environment (Teng 1986a). XSAFE inspects the soundness of the protection mechanism of a given system by launching an intrusion against the system.

Computer security is defined as the protection from misuse of a computer system, its applications and its shared resources (Neumann 1987). This includes the notions of preventing unauthorized acquisition and modification of information, thus assuring confidentiality and integrity.

Two approaches to attaining better security have been developed over the past decade. One is remedial and the other is preventative (Neumann 1978). The first approach involves evaluating the security flaws uncovered. The second approach involves designing new systems that are secure and whose security can in some way be convincingly verified.

In the last decade a few preventative models have been proposed to build "secure" computer systems. These models include the lattice model, the access matrix model, the Bell and LaPadula model, and the security kernel mechanism (Landwehr 1981).

The preventative approach is very attractive when a computer application environment requires a very high level of security since the approach designs security into the computer system. However, this approach includes the following disadvantages (Denning 1985):

- It is very expensive to develop, purchase or maintain a highly secure operating system

*This research was supported by Digital Equipment Corporation's Graduate Engineering Education Program.

**The following are trademarks of the Digital Equipment Corporation: VAX, VMS, VAX/VMS and DECnet.

- A model for a secure operating system in a network environment is not yet well defined.
- Developing systems that are absolutely secure is extremely difficult, if not impossible.

Thus the remedial approach is very desirable and affordable for computer systems and applications that require a relatively high level of security but at a lower cost.

XSAFE applies this remedial approach and inspects the security aspect of VAX/VMS system in a network environment.

VAX/VMS security has been greatly enhanced since the release of version V4.0 (Digital 1984). XSAFE assures that a given system is maintained at a relatively high level of security. Furthermore, VAX/VMS is only as secure as the user-written application programs that are layered on top of VMS. Therefore, there is a need to develop a method to detect violations of a site-dependent security policy for a given system. The method developed by XSAFE is to apply expert system technology to the domain of computer security.

An expert system approach to the security inspection of a given VAX/VMS system is strongly motivated by the following factors:

- Security experts are rare.
- A very special type of knowledge is required to discover the vulnerabilities of a computer system. An expert system, by nature, is capable of capturing and applying the expertise in a very narrow domain.
- The analysis of system security requires much knowledge of, and experience with, the VAX/VMS operating system. It is possible to express this knowledge and use it in an expert system.
- The task of adequately exploring the vulnerabilities of a VAX/VMS system would become intractable without heuristic methods for probing a computer system.
- An expert system has the potential to provide some explanation of how the security violations were discovered.
- An expert system is capable of interacting actively with a user to gather site-dependent VAX/VMS parameters.

In the computer security domain there exist very few expert systems. The proposed IDES model by SRI is implemented as a real-time Intrusion-Detection Expert System (Denning 1985). However, IDES differs from XSAFE as follows:

- IDES's approach is to detect an intrusion whereas XSAFE's approach is to launch the intrusion thus testing the soundness of the protection mechanism of a given system.
- To improve performance IDES requires a separate processor, perhaps a personal computer, to

process system audits as they are recorded. XSAFE runs only when needed and remotely in a distributed network environment.

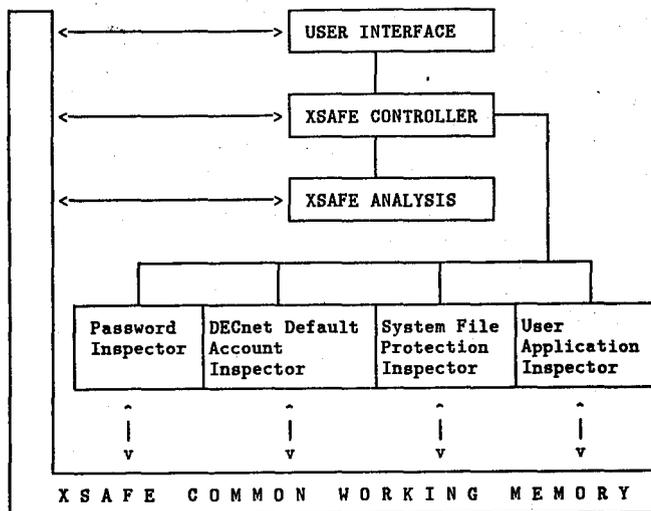


Figure 1: The Architecture of XSAFE

- IDES uses statistical knowledge whereas XSAFE uses heuristics.

2 ARCHITECTURE OF XSAFE

The architecture of XSAFE is a hierarchy of active inspection agents. XSAFE consists of a set of Security Inspectors, that contain the security-specific knowledge in the system, and a common working memory which serves as the blackboard to which all the security inspectors have access. Figure 1 shows the architecture of XSAFE.

The lines with arrows in the figure indicate read/write access to the working memory. The lines without arrows in the figure indicate control relationships among the security inspectors. Each Security Inspector consists of a set of security subinspectors with a local working memory shared among the subinspectors. Some subinspectors are further subdivided into specialists.

The architecture of XSAFE establishes a multi-level inspection structure: the XSAFE Analyst level, the Inspector level, the Subinspector level and the Specialist level. This inspection structure resembles the Blackboard architecture somewhat. However, the difference between the two architectures is that there is no direct control of the activation of each KS in the Blackboard architecture, whereas in the XSAFE architecture security inspectors are activated in a predefined sequence by the XSAFE Controller.

XSAFE uses this type of architecture for the following reasons:

- It provides a uniform structure of the common working memory. This makes it possible to integrate new security inspectors into the system easily and to develop a set of utilities applicable to all security inspectors.
- It provides a shared database where evidence of suspected weaknesses and identified security weaknesses of a VAX/VMS system are recorded,
- It allows security inspectors to use different knowledge representations and different problem-solving methods, but allows communication via the common working memory.

- It allows security inspectors to perform their security inspection independently.
- It allows higher-level security inspectors to draw conclusions from conclusions and evidence provided by lower-level security inspectors.

The XSAFE Analyst provides an integrated security view of the system under inspection. The Security Inspectors work independently of each other. The XSAFE Analyst examines those situations which the security inspectors are not able to examine due to possible interactions among inspectors. The XSAFE Analyst is activated when all security inspectors have completed their inspections. This makes the XSAFE Analyst capable of drawing conclusions from evidence and conclusions already obtained by the security inspectors.

XSAFE has the following security inspectors to examine various components of a VAX/VMS system.

- USER APPLICATION INSPECTOR. Checks if a user-written application installed on a VAX/VMS system has imposed a security threat to the underlying system.
- PASSWORD INSPECTOR. Checks if commonly known passwords can be used to log into security-critical accounts on a VAX/VMS system.
- SYSTEM FILE PROTECTION INSPECTOR. Checks if security-critical system files are properly protected.
- DECNET DEFAULT ACCOUNT INSPECTOR. Checks if the DECnet default account is set up securely.

XSAFE has been implemented in Knowledge Craft*** (Carnegie 1985), mainly because security inspection of a VAX/VMS system cannot be accomplished by any one problem solving approach. Knowledge Craft allows flexible knowledge representations, and alternative problem-solving and control strategies.

3 THE SECURITY INSPECTORS

3.1 The User Application Inspector

There has been little research and development done in the area of user-written application security in the past decade. Most research in computer security has been conducted in areas such as formal models for computer security, verification of security and computer network security. The complexity of user application security has also made it difficult to perform research in this area.

As research in formal models and verification of computer security has gradually become reality, operating systems are designed and implemented more securely. Therefore, there is a need to develop a methodology to ensure that user applications do not compromise the underlying secure operating system.

The exploration of AI techniques such as expert systems in the computer security domain provides a possible solution to the problem of user application security. This is because expert systems are capable of capturing human heuristics which are used in the security inspection of a user application.

An example of a user application could be an auto parts inventory ordering system where customers can log into the system remotely and make orders of auto parts through the application software. Another example of a user application could be a stock purchasing system where investors can look up information and prices of various stocks and make orders to buy or sell stocks.

***Knowledge Craft is a trademark of Carnegie Group Inc.

If the stock purchasing system mentioned above was not set up securely, a stock holder might gain access to the master database and change it to his advantage.

The "User Control Hypothesis" (Anderson 1972) says that security vulnerability is a function of user-controlled shared resources. In other words the less resource a user has access to, the fewer security weaknesses he will be able to discover. Hence, the major task of the User Application Inspector is to reveal those security weaknesses which are caused by inadequate control over the use of resources.

Another task of the User Application Inspector is to reveal system integrity flaws explorable in a user-written application. A user application is a piece of software developed to meet some needs of a group or groups of people.

There are four subtasks in the security inspection of a user-written application. The subtasks are carried out by the following four subinspectors and their specialists:

- The Network Communication Subinspector
- The Captive Account Subinspector
- The Login Procedure Subinspector
- The Application Program Subinspector which has an Executable Image Specialist and a Program Code Specialist

The structure of the User Application Inspector is shown in Figure 2. The lines with arrows indicate working memory accesses. The lines without arrows in the figure indicate relationships among the inspector, subinspectors and specialists.

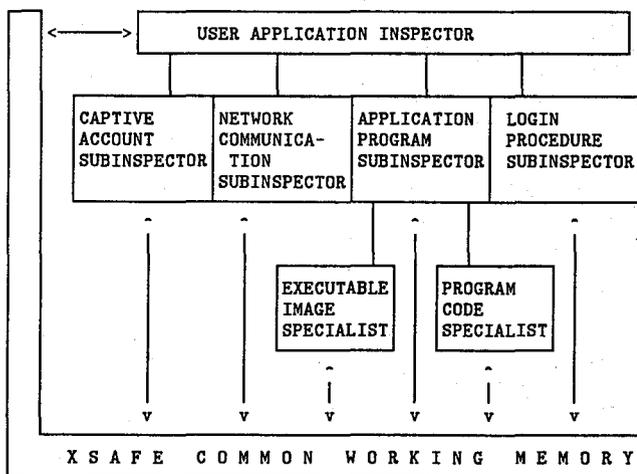


Figure 2: The Structure of the User Application Inspector

3.1.1 The Network Communication Subinspector The major task of the Network Communication Subinspector is to check for possible security problems with an application that can be used over a network. An example could be an application accessing a database on another node.

The following aspects are inspected:

- Information communicated in plain text
- Passwords being transferred over the network

A rule-based knowledge representation is

appropriate here, since the inspection is done by recognizing a situation where there is a security violation.

3.1.2 The Captive Account Subinspector The major task of the Captive Account Subinspector is to inspect an application's record in the User Authorization File (UAF) and to ensure that the setup of the account complies with the requirements for the application. Any violation of the requirements constitutes a security weakness. If an application requires that a user can not get to the supervisor level, the Captive Account Subinspector checks for certain qualifiers in the account's UAF record to ensure this requirement.

Most often a perpetrator finds his first opportunity to break out of the control of a user application by locating insecure setup of a user account.

There are two types of login restrictions that can be assigned to an account via the AUTHORIZE***Utility in VAX/VMS:

- LOGIN MODE RESTRICTIONS. Limit logins to specific types of login
- FUNCTION RESTRICTIONS. Limit types of activities of the user account

The Captive Account Subinspector gathers information about an account set-up, and about the requirements of the application via sessions of questions. Next the Captive Account Subinspector inspects the account by running rules against the gathered information. These rules detect situations where the Login mode and Function restrictions are insufficient to comply with the requirements of the applications.

3.1.3 The Login Procedure Subinspector The major task of the Login Procedure Subinspector is to discover paths that allow escape to the supervisor level ("\$\$" level in VMS) and those that break the control of a login procedure. The login procedures include both the user login command procedure and the system login command procedure.

There are various aspects of a login procedure that a perpetrator may examine to escape to the supervisor level. Some of the potential paths involve error and program abortion handling within the command procedure and the use of the DCL command INQUIRE which takes input from a user.

A procedural knowledge representation is appropriate for the Login Procedure Subinspector, since the inspection is accomplished by parsing the command procedures and searching for the presence of certain DCL commands.

3.1.4 The Application Program Subinspector A user application may compromise security because there are errors or flaws in the design, implementation, operation and maintenance of the application. Some applications may have the image of the application installed with privileges for various reasons, or the application may be running under a privileged account. These privileged applications could substantially damage a VAX/VMS system when misused by a perpetrator.

The major task of the Application Program Subinspector is to detect abuse of user functions provided by the application. For instance, a user could modify the User Authorization File if the application provides the user an editor function and is installed with SYSPRV privilege. The Application Program Subinspector inspects an application with two

***The AUTHORIZE Utility is used to maintain user accounts defined by records in a file (SYS\$SYSTEM: SYSUAF.DAT) called the User Authorization File (UAF)

security specialists: the Executable Image Specialist and the Program Code Specialist.

The Executable Image Specialist examines an application at a more conceptual level without concern about the actual coding. It reveals malicious use of functions and privileges provided directly to users and draws its conclusions based on what the functions can accomplish. Possible scenarios of abuse have been presented in (Teng 1986b).

A rule-based knowledge representation is used here. Figure 3 shows a rule in CRL-OPS for the Executable Image Specialist. The rule, here written in more natural language, records a situation where the following security violation has been discovered:

```

IF
  Installed privileges
  = (BYPASS/SYSPRV/READALL)
  AND functions of the application = (READ/MAIL/COPY)
  AND file names are specified by users or modifiable
  logical names
THEN
  Security violation = "any file including SYSUAF.DAT
  can be accessed"

```

The Program Code Specialist is activated only if the application program source code is available. It examines the coding of the application for security violations. The Program Code Specialist has not been implemented. However, it is proposed that the "Flaw Hypothesis Methodology" (Attanasio 1976) be used to detect security weaknesses in the coding of a user application program. The improvement to that methodology is the introduction of AI techniques allowing human heuristics to be captured.

3.2 The System File Protection Inspector

VAX/VMS maintains many system files that participate in various activities that keep the operating system functioning properly. These system files are critical to the security of a VAX/VMS system. The major task of the System File Protection Inspector is to check if these system files such as the User Authorization File and system startup files are properly protected. The System File Protection Inspector performs a sequence of probes to detect improperly protected system files. Hence, a procedural knowledge representation is appropriate for the System File Protection Inspector. A remote inspection of system files is accomplished by specifying the remote node name with the system file specification.

```

;-----
(P image_read_with_installed_privileges
;-----

(control ^inspector      user_application
        ^subinspector    application_program
        ^section          executable_image)

(account_inspected
  ^username_field <username>)

(application_installed_privileges
  ^username_field <username>
  ^installed_privileges << BYPASS SYSPRV READALL >>)

(application_functions
  ^username_field <username>
  ^functions_provided << FILE_SPECIFICATION_BY_USER
    FILE_SPECIFICATION_BY_MODIFIABLE_LOGICAL_NAME >>)

(application_functions
  ^username_field <username>
  ^functions_provided << VAXMAIL COPY READ >>)

-->
(Add-value xsafe_results
  ^results_from_executable_image_specialist
  ^image_read_with_installed_privileges)
) ; End of image_read_with_installed_privileges.

```

Figure 3: A Rule from the Executable Image Specialist

3.3 The Password Inspector

The major task of the Password Inspector is to detect the use of commonly known passwords. There are two types of commonly known passwords: a distributed password, which is a password that comes with the initial VAX/VMS system and is documented in the installation guide; and a guessable password which is easily obtained by some simple scheme.

The Password Inspector inspects passwords to the SYSTEM account, the FIELD account, the SYSTEST account, and any computer operational account such as the BACKUP account, as well as accounts required by the installation of software layer products such as the MRMANAGER account for the Message Router software.

The Password Inspector performs a sequence of probes to detect commonly known passwords. Consequently, a procedural knowledge representation, using LISP, is appropriate for the Password Inspector.

3.4 The DECnet Default Account Inspector

The DECnet default account is an account that is used for activating network processes on a local node. The account is like any user account on the system and has an entry in the User Authorization File.

The major task of the DECnet Default Account Inspector is to check whether it is possible to execute a program or to submit a remote batch job under the DECnet default account, to check the authorized and default privileges of the DECnet default account, and to check if the DECnet default account is grouped with other SYSTEM accounts.

The DECnet Default Account Inspector performs a sequence of probes to detect insecure setups of the DECnet default account on a given node. Hence, a procedural knowledge representation, using the DCL Command Language, is appropriate for the DECnet Default Account Inspector.

4 EVALUATION OF XSAFE

The Password Inspector, the DECnet Default Account

Inspector, the System File Protection Inspector, the User Application Inspector, the Login Procedure Subinspector, the Network Communication Subinspector, the Captive Account Subinspector, and the Application Program Subinspector with its Executable Image Specialist have been implemented. They have been running on a VAX/11 785 with 32 megabytes of primary memory. The development of the Program Code Specialist has been left for future work on XSAFE.

About 44 VAX/VMS systems on Digital's internal network were inspected by XSAFE. About half a dozen VAX/VMS system managers and VAX/VMS application developers within Digital Equipment Corporation participated in the field test process. The overall result was impressive. Some problems with XSAFE were also discovered.

Each system was tested by running the System File Protection Inspector, the DECnet Default Account Inspector, and the Password Inspector. The User Application Inspector was run when there was suitable application on the system and if the developers or the maintainers of the application permitted the inspection.

There were 153 security violations discovered by XSAFE among the 44 systems. These violations included having the User Authorization File and the Network Authorization File readable or modifiable by any user on Digital's internal network.

Compared to a security expert, the System File Protection Inspector, the Password Inspector, and the DECnet Default Account Inspector detected the kinds of security weaknesses that a security expert would find. The User Application Inspector showed that it is quite capable of handling the complexity of the security aspects of an application. In several cases, the User Application Inspector detected security violations that were missed by a security expert.

Site:	XXX
Inspector Name:	System File Protection Inspector
Number of Nodes Inspected:	44
Files Inspected per Node:	62
Average CPU Time per Node:	3.2 seconds
Average Elapsed Time per Node:	107.1 seconds

Figure 4: Performance Statistics for the System File Protection Inspector

Figure 4 shows some performance statistics for the System File Protection Inspector.

Figure 5 shows some performance statistics for the DECnet Default Account Inspector.

5 FUTURE RESEARCH AND CONCLUSION

Further investigation is necessary to gather more heuristic rule for the Executable Image Specialist. Primarily these rules should recognize a situation where a combination of certain privileges and user functions of an application constitute a security threat to the underlying VMS operating system. In addition the Program Code Specialist needs to be implemented and tested.

Site:	XXX
Inspector Name:	The DECnet Default Account Inspector
Number of Nodes Inspected:	44
Average CPU Time per Node:	6.5 seconds
Average Elapsed Time per Node:	81 seconds

Figure 5: Performance Statistics for the DECnet Default Account Inspector

We have presented an expert system approach to computer security for a VAX/VMS system. Much work remains to be done to improve and complete various

components of XSAFE. However, we feel that the expert system approach has provided a feasible solution to the problem of obtaining low-cost medium-level security for a VAX/VMS system in a network environment. The performance of the prototype expert system is very encouraging. We expect that the VAX/VMS user community will benefit from the continued development of XSAFE.

6 ACKNOWLEDGMENT

We wish to thank Steve Lipner, Secure Systems Group of Digital Equipment Corporation, for his guidance and support, and both Mitch Tseng and Tom Cervia, Applied Intelligence Systems Group of Digital Equipment Corporation, for their continued support of XSAFE.

REFERENCES

- (Anderson 1972) Anderson, J.P., *Computer Security Technology Planning Study*, ESD-TR-73-51, Vol II, pp. 89-94, ESD/AFSC, Hanscom Field, Bedford, MA, Oct. 1972.
- (Attanasio 1976) Attanasio, C.R., Markstein, P.W., and Phillips, R.J. "Penetrating an Operating System: A Study of VM/370 Integrity," *IBM System Journal*, Vol. 15, No. 1, pp. 102-116, 1976.
- (Carnegie 1985) Carnegie Group Inc., *Knowledge Craft 3.0 Reference Manual*, Vol. 1, Dec. 1985.
- (Denning 1985) Denning, D.E. and Neumann, P.G. *Requirements and Model for IDES - A Real-Time Intrusion-Detection Expert System*, SRI International, Final Report, Contract No. 83F83-01-00, SRI Project 6169, Aug. 1985.
- (Digital 1984) Digital Equipment Corporation *VAX/VMS V4.0 Release Notes*, Sep. 1984.
- (Landwehr 1981) Landwehr, C.E. "Formal Models for Computer Security", *ACM Computing Surveys*, Vol. 13, No. 3, pp. 247-278, Sep. 1981.
- (Neumann 1978) Neumann, P.G., "Computer System Security Evaluation", *Proc. National Computer Conference*, pp. 1087-1095, 1978.
- (Teng 1986a) Teng, H.S. "XSAFE: A Prototype Expert System for Security Inspection of a VAX/VMS System", *M.S. Thesis*, Computer Science Dept., Worcester Polytechnic Institute, MA, 1986.
- (Teng 1986b) Teng, H.S. "XSAFE: A Prototype Expert System for Security Inspection of a VAX/VMS System", *M.S. Thesis*, pp. 126-130, Computer Science Dept., Worcester Polytechnic Institute, MA, 1986.

The Application of "Orange Book" Standards
to Secure Telephone Switching Systems

Capt Paul D Engelman
HQ AFCC/AIZ
Scott AFB, IL 62225

Abstract

A mathematical formulation describing a telephone switching system is required to validate its operation in a multilevel communications environment. A brief description of the two major components of a telephone switch are presented, and three systems are described - two are in use and the third is postulated. A mathematical description of a security policy for each of these systems is stated. This security policy validates telephone calls between system users. A discussion of the reference monitor concept follows and provides the motivation for applying "Orange Book" standards to telephone systems. The formalities applied to computer systems are shown to apply to telephone systems, the obvious advantage is that system capabilities can be increased because of the increased trust that can be placed in the system.

Introduction

Paralleling the combination of the computer and communications fields is the merger of COMPUSEC and COMSEC into a broader discipline - INFOSEC. The ADP community quickly embraced the broader implications of INFOSEC, however, the application of computer-related INFOSEC principles to classical communication domains is still limited. Within these domains, the unspoken component of INFOSEC, OPSEC, is considered a physical security issue. The application of COMPUSEC formalities and requirements to communications systems transfers many OPSEC concerns to the system hardware and software, allowing them to arbitrate system use in a mathematically consistent and verifiable domain.

Telephone Switching

General Description

This paper discusses secure telephone systems limited to single switches within a closed environment, i.e., a command post. The system is a single computer-controlled device rather than a network of devices. This restriction simplifies the treatment and allows a focus on the key point; that telephone switching systems are amenable to a mathematical formalization identical to that performed within the COMPUSEC arena.

Within this system, the major components are:

- Station Equipment (Telephone): A transmitter/receiver which converts an acoustic signal to/from an electrical signal and provides and responds to control signalling;
- Transmission Medium: The electrical path that signals traverse; a channel or

circuit denotes an end-to-end path and the path between station equipment and the switch is a subscriber loop (or line); and a

- Switching Device: An electrical or electronic device which physically connects pairs of subscriber loops.

Overall, two distinct but interrelated modules, the switching network and the network controller, perform the telephone switching function. A set of switching devices (see above) comprises the switching network, and the network controller provides the intelligence to operate the individual switching devices.

Signalling

The sequence of events that transpires during a normal telephone conversation illustrates the relationship between acoustic and control signalling. Following is a simplified description of the control signalling necessary to support a telephone connection and the interaction between the network controller and the switching network.

1. The caller requests service from the switch by removing the telephone handset from its cradle or switch hook.
2. The network controller recognizes the "off hook" condition and sends a dial tone to the caller.
3. The caller dials the telephone which transmits the called station's address to the network controller.
4. If the called station is not busy, the network controller alerts it by sending a ringing signal.
5. The network controller provides feedback, i.e., a ringing tone or a busy signal depending on the status of the called station, to the calling station.
6. The called party accepts the call by lifting the handset.
7. The network controller recognizes the call acceptance, terminates the ringing signal and sends the calling/called party address-pair to the network controller which creates an acoustic signal path (circuit).
8. The network controller monitors the connection, releasing it when either party "hangs up."

Between steps #1 and #7, all communications between the switch and either party is control signalling. Only after step #7, when the connection is established, can acoustic signalling take place. There are different techniques for separating control

and acoustic signalling; among them, in-band signalling and above-band signalling. (Control signalling can be heard during the set-up time, the time between dialing and ringing, for a long-distance telephone call or during a conversation if the touch tone pad is inadvertently depressed.) Control and acoustical signalling use the same transmission medium, but control signals are used by the network controller and acoustic signals are routed through the switch network.

Switch Networks

Although the network controller is central to this paper, its discussion is deferred to a description of the switch network to provide a more complete description of the entire switch system. Virtually all telephone switching is circuit switching, that is, the dedication of a connection between a pair of subscriber lines for the duration of the call. The switch network is comprised of switching devices arranged to support the simultaneous connection of multiple pairs of communications channels. Modern switch matrices are broadly classified as space-division or time-division switches.

Within a space-division network, a physical link is established through the network to connect individual subscriber lines. The network appears as a matrix with each individual subscriber line connected to a single row and column of the matrix. "Shorting" the rows and columns at their intersection creates the physical connections within the network.

Figure I represents a conceptual space-division network with switching devices at each row and column intersection. In this conceptual representation, each switching device corresponds to a memory location in the network controller's memory space. To connect subscriber loops, the controller uses the originating and destination addresses to algorithmically determine the memory address of the switching device that must be "set." In the simple network in Figure I, to connect subscriber loops A and B requires the network controller to write to memory location 05.

Algorithmically,

$$M = I + N * (O - 1) \text{ where}$$

- M: Memory location of the switching device,
- I: Address of the incoming line,
- O: Address of the output line, and
- N: Number of subscriber lines connected to the matrix.

Writing a "1" to M will set the flip-flop and complete a connection between the two parties. After the call is completed, a "0" will reset the flip-flop and disconnect the lines.

Within a time-division matrix, the acoustic signals being transmitted between subscriber loops are periodically broadcast on a common bus. Each active call has an assigned "time-slot," and the subscriber loops are connected by energizing the appropriate gates when the time-slot is broadcast on the bus. The periodicity of the time-slot

sequence is dependent on the number of calls in progress.

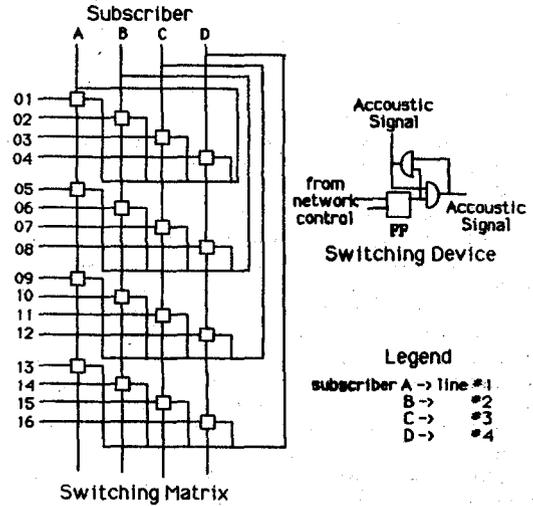


Fig. I

Figure II portrays a conceptual design of a time-division switch. The network controller places the calling/called address-pair in a circular queue, and circuitry within the network matrix cycles through the queue, using the addresses to control the opening and closing sequence of the gates. In the simple network of Figure II, as time progress:

- At T1, queue entry #1 is accessed and gates 01 and 04 are opened, connecting telephones A and D,
- At T2, queue entry #2 is accessed and gates 03 and 02 are opened, connecting telephones C and B,
- At T3, queue entry #3 is accessed and recognized as an invalid address. The network circuitry accesses queue entry #1, beginning the cycle again.

There are many different algorithms to maintain this queue, each with advantages and disadvantages. However, regardless of the algorithm, their functional behavior is identical and easily verified.

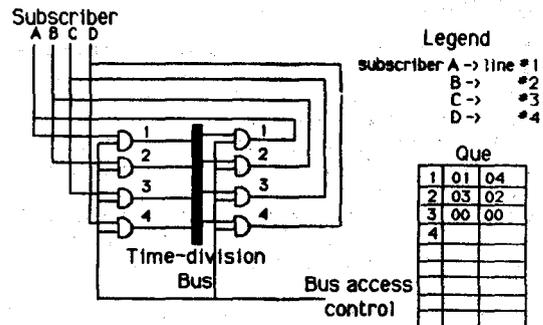


Fig. II

Unlike the time-division network, whose data stream length is directly proportional to the number of connections being supported, the time-slot interchange network has a fixed length data stream with a time-slot for each subscriber line connected to the switch. At any time, the contents of a subscriber line's

time-slot depends on the presence of acoustic signalling on that loop. The physical ordering of the calling and called parties' time-slots are interchanged when they are transferred from the input to the output data streams.

Figure III portrays a conceptual design of a time-slot interchange switch network. The input stream time-slots are "filled" and written in the same order to scratch pad memory. Input time-slot #1 is placed in memory 01, input time-slot #2 into memory 02, input time-slot #3 into memory 03, etc. Control circuitry computes the offset between the pair of time-slot positions for each connection and this dictates the access order of the scratch pad memory. A connection between subscribers A and C has an offset of 2. The network switch circuitry would copy memory 03 into output time-slot #1, memory 02 into output time-slot #2, memory 01 into output time-slot #3, etc.

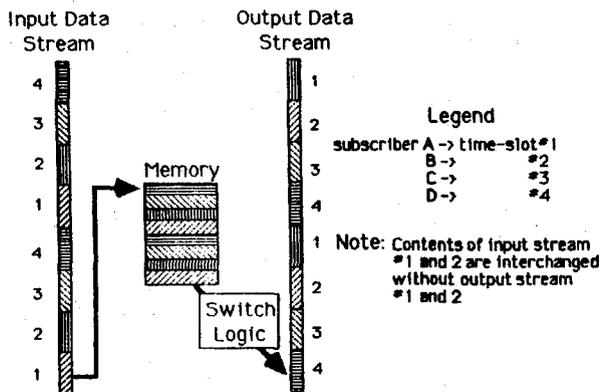


Fig. III

Although of increasing complexity, the hardware descriptions of space-division and time-division networks is relatively straightforward, as are the algorithms to translate address-pairs into control signals. Their behavior is predictable and easily verified using Boolean algebra. The concepts of a reference monitor and security kernel do not apply to the switch network because the network controller does not arbitrate circuit connections.

Network Controller

Current generation electronic switches replace wired-logic with software to provide network control functions. Referring to Figure IV, the central processor makes decisions concerning the validity of address-pairs. The controller transmits the address-pair to the internal switch network control circuitry if it determines that the address-pair is valid. The input signal device (scanner) is the device through which the network controller receives control signalling on the input side. The memory is used for program storage, and the scratch-pad memory is used for call progress information. The signal distributor on the output side serves the same function as the scanner.

Because the network controller makes all decisions concerning address-pair validity, a formal analysis of a switch network is

excluded from the following description. However, it is important to understand the relationship between the switch network, the network controller and control and acoustic signalling to fully appreciate the applicability of COMPUSEC formalities to secure switching systems.

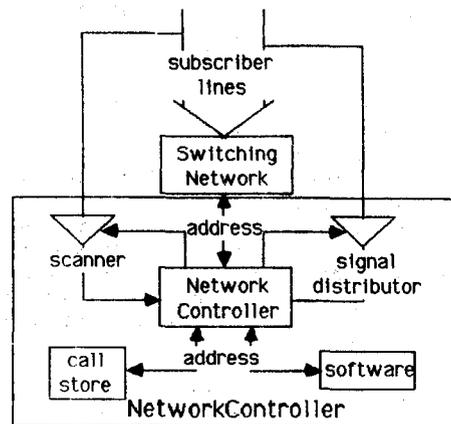


Fig. IV

Security Policy

Preliminary Discussion

The conventional COMPUSEC object, i.e., a file, does not exist within a telephone switching system. It is redefined as a subscriber line, either inactive or active in a completed circuit, that is a target for an incoming request. Similarly, in a telephone system, "read" access corresponds to monitoring an existing conversation and "write" corresponds to a broadcast message, e.g., similar to a public address system.

The improper accessing of a file or process in a computer system has its parallel in a telephone system as a misconnection. A misconnection is the connection of two subscriber loops (or a loop and a circuit for monitoring or broadcast) whose relative security levels do not meet the security policy or that violate the rules for precedence and preemption (an in-progress call can be preempted by a call of higher precedence.)

There are two sources for misconnections, an error in the algorithmic processing within the switch matrix, or a logic error in the network controller software. An example of an algorithmic error would be an unexpected overflow computing an address offset; although the network controller validated the connection, the incorrect offset will connect different loops than intended. Algorithmic errors are primarily engineering problems and are not considered in the following treatment, although, they must be considered during a system evaluation. A logic error in the controller software results in a misconnection by providing an invalid address-pair to the switch network. The source could be an error in the database that describes individual subscriber loop characteristics, or an improperly expressed condition (i.e., "(A or B) and C" vice "A or (B and C)").

The original system descriptions were prepared with an emphasis on expressing state equations. Unlike computer systems, a user can not change the security levels of a subscriber loop, so a detailed analysis of state-changes is not illuminating. As a result, only the security policy will be described, the state-change equations are very similar to those described by Bell and LaPadula.

The following models portray three generations of switching systems. The first and second are now in use; the first is simply a TEMPESTed box and the second enforces a multilevel-like security policy. The third generation is postulated. In the third generation switch, overall system security is enhanced and the system includes features not currently available because of system limitations and the limited scope of security evaluations.

First Generation

The first generation of secure switching systems can be described as a TEMPESTed box. The network controller does not enforce security. In other words, access to a telephone implies authorization to connect with any other subscriber line. Controlling system access and called party verification is an OPSEC problem. The switch supports call precedence and preemption.

Expressed in conventional set notation,

- U = {u:all subscriber lines connected to the switch network}
- S = {s:subscriber lines originating calls} These are the active entities in the system. After the circuit is established, both lines (the circuit) are potential targets for preemption and are again treated as objects.
- O = {o:subscriber lines receiving calls} These are the passive entities in the system.
- C ⊆ P(O × O) : Set of active circuits (ongoing conversations).
- L = {l:set of security and classification levels; S, TS and TSC (compartmented TS)}
- P = {p:set of precedence and preemption levels; None, Interrupt, Priority, Flash, Flash Override} A circuit created without a preemption level maps to a P (precedence) of "None." A call placed without a preemption level maps to a P (preemption) of "None."
Both L and P are connected.
- A = {a:access attributes; connect - c}
- < : An irreflexive, antisymmetric, transitive relation "less than."
- ≈ : A reflexive, symmetric, transitive relation "equivalent to."
- sl : S, O → L, a function mapping subjects and objects to their security level.
- pl : S → P, a function mapping subjects to their preemption level.
- cp : C → P, a function mapping circuits to their precedence level.
- and : Logical AND.

Any cEC that does not meet the following condition is a misconnection:

VsE, oEO,

$$s, c, o \Rightarrow \underline{sl}(s) \approx \underline{sl}(o) \text{ and } \underline{cp}(c) < \underline{pl}(s)$$

First generation systems can not enter a non-secure state. The preemption of an existing connection by a call with a lower precedence is the only possible misconnection.

Second Generation

The second generation system has the added capability of classmarking individual subscriber lines with a security level and enforcing a security policy. (Note, this security classmarking seems to be enforced as discretionary access rather than as mandatory access.) Access types are expanded to include a two-way connection, c; and one-way connections, broadcast, b and monitor, m. The switch supports precedence and preemption. The difference between this security policy and the first generation switch's is that the ordering relation on L is now "less than or equal to" rather than an equivalence. In conventional set notation,

U, S, O, C, L, and P : As described above.

A = {a:access attributes; connect - c, broadcast - b, monitor - m}

< : As described above.

≤ : A reflexive, antisymmetric, transitive relation "less than or equal to"

sl, pl, and cp : As described above.

cl : C → L, a function mapping circuits to their classification level, the $\text{Min}(\underline{sl}(o_1), \underline{sl}(o_2))$.

Any cEC that does not meet the following condition(s) is a misconnection:

VsE, oEO,

$$s, c, o \Rightarrow \underline{sl}(o) \leq \underline{sl}(s) \text{ and } \underline{cp}(c) < \underline{pl}(s)$$

$$s, b, o \Rightarrow \underline{sl}(s) \leq ((\underline{sl}(o), \underline{cl}(c))) \text{ and } \underline{cp}(c) < \underline{pl}(s)$$

$$s, m, o \Rightarrow (\underline{sl}(o), \underline{cl}(c)) \leq \underline{sl}(s)$$

The rationalization for this policy is as follows. A subscriber receiving a call does not receive an indication of the originating line, only the incoming line. The called party must insure that the classification level of the conversation does not exceed the security level of his circuit. The call originator is responsible for knowing the security level of the called party's line and to restrict the conversation appropriately.

A security policy allowing the connection of a higher to a lower security level may seem contrary to "normal" computer operations, and obviously provides a covert channel. The nature of a telephone system is two-way communications; if this two-way connection between security levels is considered from an integrity viewpoint rather than a strictly security viewpoint, it is more palatable. The alternative is a rigid enforcement of security levels, which at this point, seems extreme.

Third Generation

The third generation secure switches have expanded capabilities and features and can reduce the OPSEC problem. However, there is an increased demand on the system users. In this system, subscriber lines are not passive entities, but active elements; at any time, specific users are associated with specific lines. Essentially, each user must log-in before using the system, and the switch, through access control lists (ACLs), can enforce both system and specific circuit use.

A circuit request is redefined as an interprocess invocation, \downarrow , to emphasize that telephone calls are now between user/circuit combinations rather than simply between circuits. Conventional objects now exist, these are user voice mailboxes. The voice mailbox is used if a called party is busy, and the calling party can not, or chooses not, to interfere with the call. Security control of mailboxes is identical to that of any object in a secure computer system, the contents are digitized voice messages rather than ASCII strings.

The policy for voice communications is similar to the above, with the additional requirement that an individual must have specific access (discretionary access) to specific circuits.

U, C, L, and P : As described above.
S = {s|subjects; system users}
O = {o|objects; user voice mailboxes}
 $I \subseteq P(S \times S)$: Set of possible interprocess connection.
A = {a|access attributes; interprocess communication - \downarrow (similar to \underline{c} above), broadcast - \underline{b} , monitor - \underline{m} , read - \underline{r} , write - \underline{w})
<, \leq : As described above.
 \underline{s}_1 , \underline{p}_1 , \underline{c}_p , \underline{c}_l : As described above.
user : S \rightarrow U; a function mapping subjects to users.
acl : S, O \rightarrow P(U \times A); a function mapping subjects and objects to <user,A> pairs.
name : U \times A \rightarrow U; a function selecting the user component of an ACL entry.
mode : U \times A \rightarrow A; a function selecting the access component of an ACL entry.

As above, any $I \in I$ that does not meet the following condition is a misconnection:

$\forall s_1, s_2 \in S,$
 $s_1, \downarrow, s_2 \Rightarrow \underline{s}_l(s_2) \leq \underline{s}_l(s_1) \text{ and}$
 $\underline{c}_p(c) < \underline{p}_l(s_1) \text{ and}$
 $\exists \text{acl} \in \underline{acl}(s_2),$

where

name = user(s_1) and mode(acl) = \downarrow

The conditions for broadcast and monitor follow logically, as do the conditions for reading and writing to mailboxes. The use of ACLs allows the network controller to arbitrate access to a finer granularity than within the second generation switch, a capability particularly desirable for compartmented information. Because access

control is limited and only authorized users (validated by the switch) can make and receive calls on specific circuits, the need for OPSEC-oriented authentication procedures (i.e., voice recognition) is eliminated. Voice mailboxes are allowed, because they are simply files on a computer and easily proven secure.

APPLICATION OF THE "ORANGE BOOK" STANDARDS

Reference Monitor

The concept of a reference monitor is equally applicable in a secure computer system and a secure telephone switch. Admittedly, there is a distinct difference between a one-direction computer message and a telephone conversation, but the reference monitor's function is to enforce a formalized security policy, not monitor information flow. Neither is a specific security policy implied, nor is the implementation mechanism relevant.

The reference monitor for a telephone switching system must satisfy three logical properties: 1) all connection requests must be monitored and the security policy (reflected in the switch database) enforced, 2) the reference monitor is unmodifiable by common users, and 3) it has provable behavior.

From the above discussion of telephone switches, it is obvious that the network controller displays the first two properties, and state-change functions have been derived elsewhere that demonstrate a technique for verifying network controller behavior. The behavior of the switch network can also be verified. Unfortunately, there appears to be little on-going effort to formally prove consistent, secure behavior within the current secure switching systems - the purpose of this paper is to motivate a detailed investigation into that area.

Orange Book Standards

The fundamental computer security requirements described in the Orange Book are 1) an explicit and well-defined security policy, 2) access control markings associated with system objects, 3) identification of individual subjects, 4) system auditing and protection of information on security related actions, 5) a system security enforcement mechanism capable of being analyzed, and 6) a continually protected security enforcement mechanism. The reference monitor concept is inherent in these requirements. The security kernel is the hardware and software realization of the reference monitor concept.

The switch architecture meets all the above criteria. A security policy has been stated, and is reflected in the switch database. Switch operation requires control markings of system objects (subscriber lines). Although lacking in the first and second generation switches, the third generation switch has provisions for uniquely identifying individual subjects. Current generation switches audit all connections and all maintenance actions. Using the methodology outlined in Bell and LaPadula and elaborated

on above, the switch enforcement mechanism can be analyzed. And finally, the average user can not directly access the network controller software, which continually protects and isolates the security enforcement mechanism.

CONCLUSION

The initial motivation for applying the Orange Book standards to secure telephone systems was an analogy between the telephone system's physical components and a simple, multi-user computer system. The network controller is analogous to the computer (CPU, memory, etc.), the switch network is analogous to the front-end processor and the telephone instruments are analogous to the terminals. Further investigation into switching systems reinforced the analogy; current time-slot interchange switches even packetize the voice traffic.

The primary difference between telephone systems and computer systems is that telephones transmit voice traffic (either as analog or digital signals) and computers transmit ASCII characters. Also, the voice signals never enter the network controller (or computer). However, if the telephone system is viewed as a "black box," information enters, is rerouted and exits, just as in a computer system. The same basic mathematical formalisms and evaluation criteria apply.

The advantages of using the Orange Book evaluation criteria are manifold. The most obvious are the ability to increase the capabilities and services of the telephone system with a significant measure of confidence, and the use of a consistent, well-defined and understood evaluation criteria for system certification.

ACKNOWLEDGEMENTS

The author gratefully acknowledges the patience and assistance of Russ Housley, Senior Systems Engineer, Xerox Special Information Systems, in the review and editing of this paper.

Bibliography

1. R. F. Rey, Technical Editor Engineering and Operations in the Bell System, AT&T Bell Labs, 1973
2. D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations," MITR 2547 Vol 1, Nov 1973.
3. K. J. Biba, "Integrity Considerations for Secure Compute Systems," MITR 3153, Apr 1977.
4. R. A. Gove, "Modeling of Computer Network," In Proceedings of the 8th National Computer Security Conference, 1985
5. DoD 5200.28-STD, DoD Trusted Computer System Evaluation Criteria, Dec. 1985.

THE NATIONAL COMPUTER SECURITY CENTER
TECHNICAL GUIDELINES PROGRAM

Phillip H. Taylor
Chief
Technical Guidelines Division
National Computer Security Center
9800 Savage Road
Ft. George G. Meade, MD 20755-6000
(301) 859-4452

Introduction

Background

The Technical Guidelines Division of the National Computer Security Center produces, and supports others who produce, Computer Security technical guidelines publications. The purpose is to provide a national computer security literature base that distributes computer security knowledge and techniques, instills an accepted computer security terminology, and applies research to practical problems of computer security.

The National Computer Security Center (NCSC) has working relationships with many other organizations. Their support and assistance is critical to the overall success of the technical guidelines program. The Technical Guidelines Division is a service bureau in many ways. We produce work required by our customers, prioritized according to our customers needs. We bring together the wisest people we can find, whether from the private sector, from a university, or from another government agency. Our coordination efforts are wide and constant. The goal is to produce the best and most usable technical guideline possible. We want to produce guidelines that are easy to read and understand, unambiguous, representative of all sectors, and helpful.

Purpose

The purpose of this article is to provide an update on the status of computer security technical guidelines. Also included is an explanation of the levels of guidelines that exist and how they interrelate, how the requirements process works that kicks off new computer security technical guidelines projects, how the projects are done, who is involved, and what the future looks like for Computer security technical guidelines.

Scope

The scope of this article includes the technical guidelines work done at the NCSC and its contributors in the private and civil sectors and the academic community. The project status summary (given as of the date of the presentation) includes the purpose of the individual projects such as the Trusted Network Interpretation, the Trusted Database Interpretation, the Trusted UNIX Design effort, and "How To" guidelines.

The NCSC and the Technical Guidelines Division

Why Guidelines?

Guidelines are not the dictates of a government agency! They are not intended to limit, control, or in any way constrict thinking to preset hard ideas. They exist to document a common set of fundamental principles of computer security. They also are intended to serve as a source of common language and approaches to help communicate about and implement computer security.

Specifically, the guidelines support education for vendors, users, and evaluators. They greatly reduce the start-up time needed when beginning work on computer security. The guidelines serve as tools themselves, and suggest other tools for the evaluation and implementation of computer security. Certainly one of their most valuable uses is that they spread the gospel of computer security and expand the cadre of experts. The National Computer Security Conference and its growth document the rapid expansion of folks who take computer security seriously. The guidelines are focal points of knowledge concerning computer security on specific architectures such as trusted networks and subsystems.

What Has Been Happening?

The Technical Guidelines Division at the NCSC might be more recognizable as the old Standards Division. Since our brothers at the National Bureau are tasked with writing standards and the NCSC writes technical guidelines we have changed the organization title.

As of June 1987 the NCSC had produced 12 guidelines, was working on nearly 30 additional guidelines, and had identified and prioritized more than 20 additional.

Note that the large "IDENTIFIED" area in Illustration 1 is disproportionately larger than the number in it when compared with the other slices of the pie. This is because there are many more publications that are yet to be identified. Illustration 2 lists those projects that have been published by the NCSC as of June 1987.

Program Specifics

Program Structure

The technical guidelines created at the NCSC are part of an overall national framework for computer security technical documentation. This national framework has been devised to assure that the many people and organizations interested in computer security are represented, the required technical publications are produced, and duplication of effort is minimized.

The technical guidelines program has two levels. They are the Evaluation and Design Level and the Support Level. The Evaluation and Design Level is a level of high abstraction and truly represents the cutting edge of computer security technology. In Illustration 1 the top level represents the fundamental principles of computer security. These fundamental principles are derived from the requirements of computer security policy. The second level is the criteria and interpretations level. These two top levels are the Orange Book level in that the original Orange Book was made up of the fundamental principles of computer security and the criteria for the evaluation of a stand-alone operating system.

Now the technical guidelines program is creating new criteria and interpretations for the evaluation and design of trusted networks, subsystems, database management systems, distributed systems, and tactical and embedded systems. The criteria and interpretation publications detail the features

required in a specific architecture for it to be trusted at defined levels. The criteria and interpretations also contain metrics that designers, evaluators, and users can apply to the features in an architecture or system to gauge if the required computer security features are indeed included and work as intended.

There are technical "How To" books and administrative "How To" books. The technical "How To" books are intended to flesh out how to use the metrics and features discussed in the criteria and interpretation level guidelines for the different architectures. For example, how does one do configuration management for a trusted network? The administrative "How To" books will look at non-technical assistance such as "How to Begin an Evaluation with the National Computer Security Center".

The most important project recently underway at the NCSC is the Trusted Network Interpretation. This guideline is essential in that the nation is building systems based on network architectures and will expand these efforts in the future. There is a large effort now towards building secure networks. The trend towards building architectures with widely distributed workstations, servers, and users creates special needs for increased security. In fact, we compare working on the Trusted Network Interpretation with a mini manned space shot -- the work is very complex and terribly difficult, every step is completely new ground, and there is absolutely no room for error.

At the Support Level of the technical guidelines program there are many players. For example the National Bureau of Standards contributes much of its Federal Information Processing System Standards to this level. The NCSC's project on a Trusted UNIX Design is also at this level.

The Requirements Process

The requirements process that establishes a technical guideline as a project is not complex. Initially the project planning process was driven by the needs of the evaluators as defined in the Management Plan that governs all evaluations at the NCSC. The Management Plan specified guidelines to compliment the work of the evaluators and system designers. When the Orange Book was the sole criteria and stand-alone systems were the only systems being evaluated this was simple enough. But the complexity introduced by beginning evaluations on trusted networks, subsystems, and in the future a broad range of system

architectures mandated more structure. As discussed earlier the different levels of technical guidelines have evolved and now each level of that structure has requirements.

Now, too, there are new people and organizations involved in computer security and they have their own special needs. The "How To" books become increasingly important as more interested persons get involved - many of whom have limited experience with Computer security and evaluations. The expanding number of architectures being evaluated requires more criteria and interpretations to be written.

Now we must insure that all have a chance to participate in the requirements process. We have begun writing articles for publication that will reach the new and old players and invite their suggestions for new guidelines or recommend changes to old ones. This paper is part of that process. I invite you to forward your ideas and suggestions to the Technical Guidelines Division at the NCSC. Suggestions will go to the Technical Guidelines Review Board which has been established for this purpose. All suggestions will get responses.

Setting Priorities

Obviously a problem is evolving. Where are the resources to produce all of the technical guidelines? Which guideline is most urgently needed? Who has the most critical need for a technical guideline to address their problem? Certainly there are few experts around to lead these efforts. The growing number of technical guidelines requirements, the success of computer security as reflected by the awakening of a national will for computer security, and the staggering increase in the number of systems being built with computer security in mind tax the previously serial production of guidelines.

Now clearly we must carefully assign priorities to projects based on national need and the greatest impact. The priority list is developed by the Technical Guidelines Division after gathering inputs from many sources. At the end of this talk we will give you a survey to fill out that will help us in our priority planning as well as in our quality assurance program. This is one of a number of efforts we have underway to determine what our priorities should be.

How Projects are Done

The development process that results in a guideline can be quite varied depending on the technology being documented. Some of the factors that determine the process are who is

going to lead the work, who is going to do the work, when is the guideline needed, and how much money is available?

In general the development process begins with a search of the existing literature and an exploration into who knows the most about the subject. The issues that must be addressed and solved are determined and solidified. Then a dialogue is initiated, usually in person, by phone and through DOCKMASTER. All of this interaction results in an issues paper that is intended to serve as the basis for a guideline. The issue paper is also the strawman that generates comments from interested parties. From the comments the scope and specific intent of the future guideline can be determined. The project manager can generate a tasking plan that will assign a whole document or parts of a document to those who will be the authors. Then the authors synthesize the total research to that point into a draft guideline. The drafts are published for review and comment; first to a small group of very knowledgeable reviewers, and then later to a more general audience. At last, the guideline will be published.

Project Status

Illustration 5 shows the technical guidelines program timelines as of June 1987 for the projects coordinated by the NCSC. The project is listed in the year that it will be completed. Note that some projects are underway now even though they are not scheduled for completion until FY90. Clearly these projects are tough and require long lead times for research and maturation. The asterisks indicate that the project is underway.

Future Projects

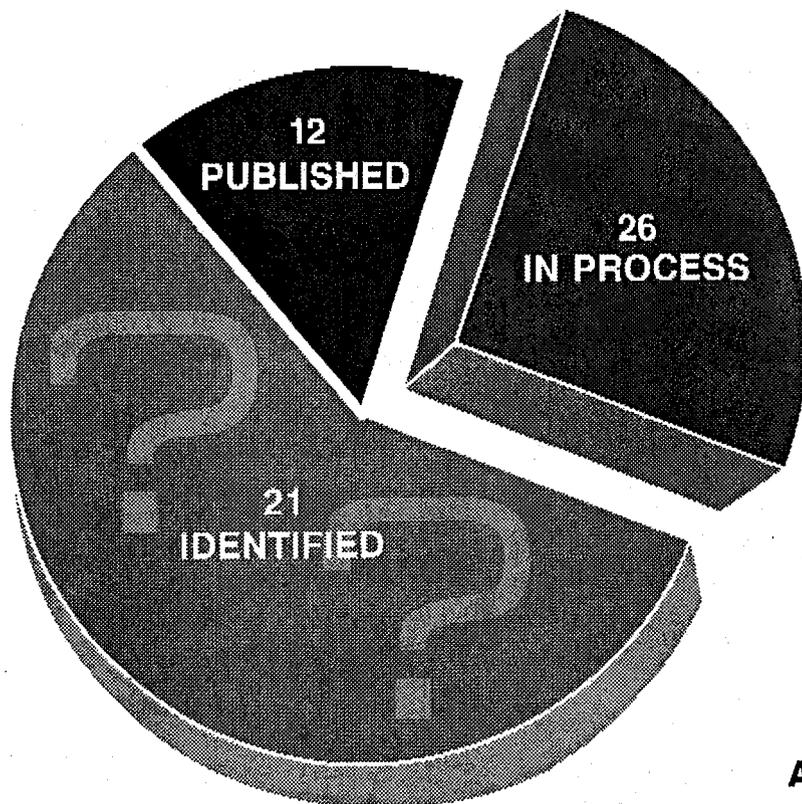
The guidelines requirements through the year 1990 are pretty clear. Obviously, some guidelines are much more difficult than others. Among the most difficult are the tactical systems guidelines and the embedded systems guidelines. While they are not scheduled for publication until the 1990 period we have begun our investment in them and we are in the literature research phase of the preparation cycle.

Illustration 6 shows that the greatest number of publications underway in the 1988 through 1990 time period are in the "How To" series. These are the guidelines that the evaluators, designers, and users will use most in their daily work. The numbers of "How To" books will grow significantly as the criteria and interpretation level guidelines are produced.

Conclusions

The technical guidelines program has a key role in the development of computer security. It provides the library for computer security work. It gives us a common language and a common view of the computer security world. Even if we disagree, it provides us something to disagree about with reference points and metrics for discussion. It is in the technical guidelines that we codify our understandings of the science and it is here that we merge our ideas about how to secure new architectures. This is where we look ahead.

TECHNICAL GUIDELINES



AS OF 15 APR 87

ILLUSTRATION 1

WHAT HAS BEEN WRITTEN?

- TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA - 1983
- TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA - 1985 (DoD-STD)
- COMPUTER SECURITY REQUIREMENTS - 1985
- RATIONALE BEHIND THE COMPUTER SECURITY REQUIREMENTS - 1985
- PASSWORD MANAGEMENT GUIDELINE - 1985
- NETWORK WORKSHOP PROCEEDINGS - 1985
- MAGNETIC REMANENCE - 1985
- TRUSTED NETWORK EVALUATION CRITERIA - 1985 (DRAFT)
- "COMPUSECese" COMPUTER SECURITY GLOSSARY - 1985 (EDITION 1)
- PERSONAL COMPUTER SECURITY CONSIDERATIONS - 1985
- DATABASE MANAGEMENT SYSTEMS SECURITY WORKSHOP PROCEEDINGS - 1986
- A GUIDELINE TO OFFICE AUTOMATION SECURITY - 1987

ILLUSTRATION 2

DESIGN AND EVALUATION LEVEL

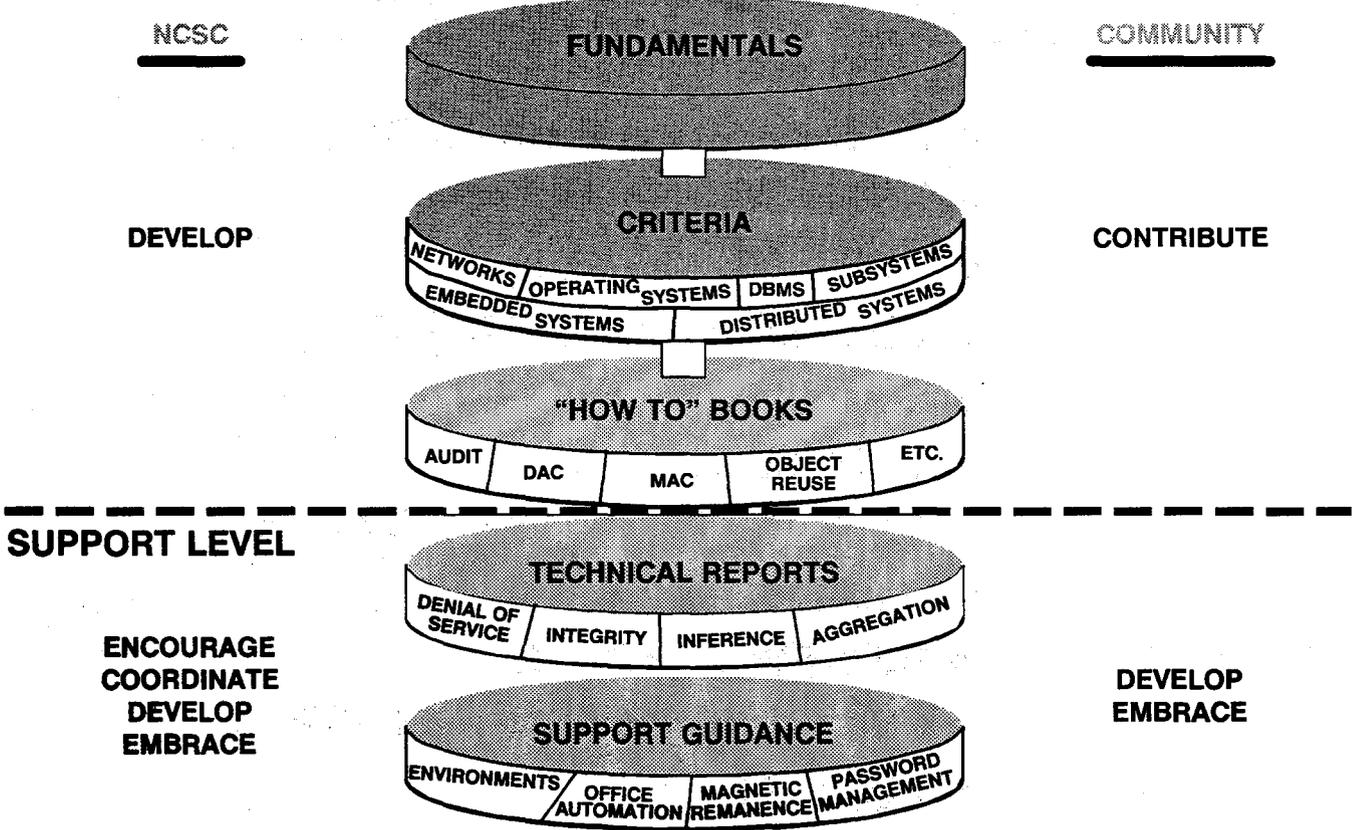


ILLUSTRATION 3

DEVELOPMENT PROCESS

- **LITERATURE RESEARCH**
- **DETERMINE ISSUES**
- **INITIATE DIALOG**
- **DEVELOP ISSUE PAPER**
- **SOLICIT COMMENTS**
- **DEFINE SCOPE**
- **TASK FCRCs/CONSULTANTS**
- **SYNTHESIZE**
- **REVIEW/COMMENT**
- **TECHNICAL REVIEW BOARD**
- **PUBLISH**

ILLUSTRATION 4

TECHNICAL GUIDELINES PROGRAM DEVELOPMENT SCHEDULE

FY87	FY88	FY89	FY90
<ul style="list-style-type: none"> *TRUSTED NETWORK INTERP *NCSC ORANGE BOOK *CONFIGURATION MGNT *CIVIL SECTOR ENVIRON *WORKING WITH THE NCSC *DAC *AUDIT *REVISED MAGNETIC REM *QUALIFIED PRODUCTS LIST *GLOSSARY CRITERIA REVIEW BOARD-CTD OFFICE AUTOMATION-CTD 	<ul style="list-style-type: none"> *SUBSYSTEMS INTERPRET *TRUSTED UNIX DESIGN *SYSTEM ARCHITECTURE *DESIGN DOCUMENTATION *LABELING *COVERT CHANNEL ANALYSIS *TRUSTED FACILITIES MANUAL *DAA ACCREDITATION GUIDE *MAC *NETWORK TESTING GD LN SSO GUIDELINES SYSTEM INTEGRITY SECURITY TESTING PRODUCT DISTRIBUTION SECURITY FEATURES GUIDE TRUSTED PATH PRIVATE SECTOR ENVIRON *DECLASSIFICATION SOFTWARE *PRODUCT ACQUISITION GUIDE 	<ul style="list-style-type: none"> *DBMS INTERPRETATION TEST PLANS & DOCUMEN DESIGN SPECS & VERIF HDW/FMW VERIFICATION SOFTWARE VERIFICATION SECURITY MODELS AIS INSPECTION GUIDE AIS STANDARD PRACTICE OBJECT REUSE TRUSTED FACILITY MGNT TRUSTED RECOVERY ELECTRONIC MAIL PRIVACY 	<ul style="list-style-type: none"> DISTRIBUTED PROCESSING IDENT. & AUTHENTICATION *TACTICAL SYSTEMS *EMBEDDED SYSTEMS INSIDER THREAT SECURITY MODEL INTERPRET

* INDICATES PROJECT UNDERWAY

AS OF 15 APR 87

ILLUSTRATION 5

**SUPPORT
DOCUMENTATION**

HOW TO BOOKS

CRITERIA

**TECHNICAL
REPORTS**

TRUSTED UNIX DESIGN
CIVIL SECTOR ENVIRONMENTS
WORKING WITH THE NCSC
DAA ACCREDITATION GUIDE
OFFICE AUTOMATION
SSO GUIDELINES
PRIVATE SECTOR ENVIRON.
DECLASSIFICATION SOFTWARE
REVISE MAGNETIC REMANENCE
QUALIFIED PRODUCTS LIST
CRITERIA REVIEW BOARD
GLOSSARY
PRODUCT ACQUISITION GUIDE
AIS INSPECTION GUIDE
AIS STANDARD PRACTICE
ELECTRONIC MAIL PRIVACY

SYSTEM ARCHITECTURE
DESIGN DOCUMENTATION
CONFIGURATION MANAGEMENT
TEST PLANS & DOCUMENTATION
LABELING
COVERT CHANNEL ANALYSIS
TRUSTED FACILITIES MANUAL
DAC
MAC
NETWORK TESTING GUIDELINES
DESIGN SPECS/VERIFICATION
AUDIT
SYSTEM INTEGRITY
SECURITY TESTING
PRODUCT DISTRIBUTION
SECURITY VERIFICATION
SECURITY MODELS
IDENT AND AUTHENTICATION
OBJECT REUSE
TRUSTED FACILITY MANAGEMENT
TRUSTED RECOVERY
SECURITY MODEL INTERPRETATIONS

TRUSTED NETWORK INTERPRET.
NCSC ORANGE BOOK
SUBSYSTEMS INTERPRETATIONS
DBMS INTERPRETATION
DISTRIBUTED PROCESSING
TACTICAL SYSTEMS
EMBEDDED SYSTEMS
INSIDER THREAT

AS OF 15 APR 87

ILLUSTRATION 6

GETTING ORGANIZATIONS INVOLVED IN COMPUTER SECURITY:

THE ROLE OF SECURITY AWARENESS

by Elizabeth Markey

Chief, Policy and Awareness Division

Office of Information Systems Security

Bureau of Diplomatic Security

U.S. Department of State

Objectives of Presentation

To learn how to get organizations aware and involved in computer security through on-going training and awareness programs aimed at employees at all levels.

Background

U.S. Department of State automated information systems are used at over 150 diplomatic and consular posts worldwide for word processing, financial disbursements and controls, personnel functions, issuance of passports and visas, and other important management functions. Because of the sensitivity of many of these systems, their security is being given increasing emphasis by State Department management. Responsibility for automated information systems security has been assigned to the Office of Information Systems Security (ISS) of the Bureau of Diplomatic Security, but ISS cannot do the job alone. The effectiveness of the systems security program depends to a great extent on the participation of other elements of the State Department, particularly managers, line security personnel, and users.

Currently ISS is conducting a series of seminars and briefings aimed at employees at all levels which include the following:

- 1) A 2-hour briefing for Executive Directors of our regional and functional bureaus in the Department;
- 2) A 4-day seminar for Regional Security Officers (RSOs), who are responsible for security at our overseas embassies; and
- 3) A 1-2 hour briefing for all new employees with the Department.

The objective is to ensure that all employees are up-to-date on computer technologies used throughout the Department, and have the information needed to participate effectively in the Department of State computer security program.

Basic Messages Conveyed

These systems security seminars and briefings are tailored to meet the varying levels of knowledge, experience, and responsibilities of all employees.

The briefings for Executive Directors stress the potential consequences arising from the lack of adequate protection of the organization's telecommunications and automated information systems resources, and the commitment of the organization to protect automated systems resources. Executive

Directors are briefed on current National and State Department system security policies and standards, as well as potential threats and vulnerabilities of our systems. The main objective here is to ensure that computer security in the Department first and foremost receives support from top management.

The 4-day seminars for Regional Security Officers contain much more in-depth information. For example, officers learn enough about how Department of State computer systems function to be able to ZAP a password file, browse a user's directory of files, and monitor the activities of the System Administrator. The goal is to give the officer a good understanding of the technical aspects of automated information systems to know where and why security vulnerabilities occur, and how to detect and correct them. Second, this seminar includes four "hands on" lab sessions using a Wang VS computer system. These sessions enable each officer to try out the ideas presented in the first part of the seminar at a computer terminal. The officers learn word and data processing capabilities, password administration, and how to spot potential weaknesses in the system. In the third part of the seminar, the officers are briefed on current State Department automated information systems, security policies and standards, and potential threats to the systems. The emphasis here is on the practical application of the first two parts of the seminar to the actual conditions which security officers will encounter at overseas State Department posts.

Briefings are also held for all new employees before they begin their employment with the Department. Virtually all of these new employees will become users of our automated information systems. For the most part, these briefings stress, in non-technical terms, the threats and vulnerabilities of departmental systems and how computer security impacts them directly. We also give users instructions on how to protect the integrity of the computer and the information that goes into and out of it. The focus here is on "do's" and "don't's". Finally, we stress why the user should be concerned with good security practices and how they should react to potential problem situations.

Each seminar and briefing has been carefully structured to support our overall objective: continued effective participation by all employees in the systems security program. Evaluations by participants, and later feedback confirm that these briefings and seminars are meeting this objective.

Lessons Learned

In 1987, automated information systems security

must be part of every employee's job. The computer security unit in a large organization cannot hope to cover all bases by itself. The experience with the State Department systems security seminars and briefings has shown that employees at all levels can participate actively in supporting systems security goals. But there are two important prerequisites. Systems security policy and procedures must be carefully delineated. It is essential that basic policy objectives, and specific security procedures be constructed to support the mission of the organization, and that the policy has the support of line organizations. This requires all concerned parties to have a hand in the policy review and approval cycle. Likewise, the responsibilities of each unit of the organization must be well defined. The Office of Information Systems Security has followed this path in the publication of four detailed automated information system security standards which have been adopted by the Department of State.

Although these points are generally understood, training and awareness activities may not always receive the attention they deserve. Computer operators and technicians may feel that systems concepts are too complex to be grasped by "non-technical" people. The State Department experience has shown that this is not so. Of course, training goals must be set realistically. An analysis of the published security responsibility assignments will show exactly what each employee needs to know to do the job assigned to them. If the content of the training is sharply focused on these needs, it will be apparent to the audience, and they will be motivated to apply themselves and absorb the material. Once they gain confidence in their ability to deal with computer security matters, they will become active participants in the automated information systems security program.

The development and conduct of computer security training and awareness activities is not a simple task. A substantial investment in time by the systems security unit is required. However, the resulting contributions by the organization's employees will repay the effort many times over. Managers, line security people, and end users with the proper training and support can augment the eyes and ears of the systems security unit, contribute expertise in physical security and investigation of security incidents; in short help to build a team effort to strengthen automated systems security.

The Computer Security Training
Base of 1985

Eliot Sohmer
National Computer Security
Center

In August 1985 the Director of the National Computer Security Center (NCSC) established a special task force consisting of six senior Center personnel. The task force was the result of the Director's recognition that there was no established curriculum of computer security (COMPUSEC) courses and that Center personnel possessed a wide range of capabilities and vastly different knowledge bases. The task force's job was to assess the situation and make recommendations to the Director for corrective action.

The task force, led by Eliot Sohmer, Chief of the Office of Product Evaluations and Technical Guidelines within the NCSC, issued its final report and recommendations on 24 October 1985. The recommendations of the report were accepted and are now being implemented by the Center.

Ultimately, the training laid out in the plan will be made available to anyone interested in receiving it. We will start by training Center personnel. Our plan is to fit the courses together into a coherent whole so that the material "flows" from concept to concept. We will then video tape the training and make the tapes available to other government agencies, universities, and vendors.

The task force's final report identified nine categories of Center personnel ranging from product evaluators to research and development specialists to clericals (see enclosure 1). We included clericals and administrative assistants to increase their awareness of COMPUSEC issues so all Center personnel could work as a team in this adventure called the "COMPUSEC revolution."

The task force identified eighteen courses we believed were needed. Of these, twelve were non-technical and six were

technical (see enclosure 2). We then built a matrix that enabled us to recommend to the Director which of the nine categories of employees should take which of the eighteen training modules (see enclosure 3).

The task force also produced a summary of what we thought would be appropriate information to include in each module (see enclosure 4). In so doing, we gave a curriculum committee a head start in putting together the courses.

Since the final report was issued, the Office of Technical Support within the NCSC has taken the initiative and developed or supervised the development of most of the non-technical courses. The Center is now in the process of developing all of the technical courses. A seventh course, one on penetration, has since been added to the technical offerings.

Finally, the task force also developed a suggested "road map" detailing a logical sequence in which personnel could be guided through various parts of this program (see enclosure 5).

I believe the task force's work and the subsequent effort within the Center to implement its recommendations will have long-term, significant effects on the National Computer Security program. The training material developed will help many sources such as universities, government agencies, computer manufacturers, and the evaluation community to develop consistency in their approaches to COMPUSEC.

THE NINE CATEGORIES OF CENTER
PERSONNEL: (Enclosure 1)

- I. Product evaluator
- II. System evaluator
- III. R&D specialist
- IV. Technical implementation specialist - an engineer working on implementing computer security, such as BLACKER personnel

V. Manager - some individuals may be required to take modules identified for this category, as well as for another category (example: a supervisor who is also a systems evaluator)

VI. Trainer - individual who works with Center education and awareness programs

VII. Support - basically, non-technical personnel who do not fall into one of the other categories (e.g., C13 and C23 personnel)

VIII. Administrator - individuals in personnel administration and other primarily staff functions

IX. Clerical

LIST OF TRAINING TOPICS (Enclosure 2)

Non-Technical Courses:

1. Orientation to Computer Security Issues:
Standard terminology and basic concepts, Lines of Defense, Threat and Vulnerability
2. Center Organization and Mission:
Center Organization and Major Activities, The Center Within NSA, Within DoD and Within the Federal Government
3. Our Fundamental Beliefs and Policy...The Catechism
4. Policy, Directives, Regulations, and Legalities:
NTISSC, SAISS roles;
Other directives and regulations as appropriate
5. Fundamentals of Classification:
Covernames, codewords, compartmentation, etc.
6. Ethics and Responsibility of Center Personnel:
Computer Usage (in general and as an individual)
Government employees' responsibilities
7. Measuring Computer Security:
Introduction to Criteria, Standards, and Guidelines
8. Criteria Part II:
C1-B1; B2; B3-A1
9. Evaluating the Environment- An Overview
10. Risk Management
11. Administration of Computer Security in an Organization:
12. COMSEC Overview

Technical Topics

13. Architectures:
Implementation issues, Technical Credibility of an Implementation, Show how specific architectures either support Criteria or not
14. The Criteria (Technical Version):
Philosophical/policy underpinnings;
Derivation of requirements from "first principles";
Structure of Criteria;
Main elements of each division/class (object reuse, mandatory controls and labeling, formal methods);
Inter-dependence of requirements;
Relationship of documentation required for above
15. Theoretical Foundations Introduction to Logic Security Policy Modeling:
Basic concepts - modeling, access control mechanisms, etc.;
Bell-La Padula, information flow, non-inference, multilevel objects; Finite state machines
16. Model Interpretation:
Translation of higher levels of abstraction into hardware/software design;
Assurance that implementation enforces rules of policy model
17. Correctness:
Specifications
Metatheorems
Implementation Correctness
Formal Semantics of Programming Languages
Predicate Transformation
Correspondence mapping
Issue of formal, unambiguous specification languages;
Issue of information flow (covert channel analysis) and invariant analysis;
Implementation Capabilities and limitations of technology;
Tools developed to apply

theory; fundamentals of Hoare logic

18. Evaluation Theory and Practices:

Examination of theoretical underpinnings of the three major classes of the criteria and how modeling and assurance concepts are embodied in each.

TRAINING TOPICS/PERSONNEL CATEGORIES
(Enclosure 3)

1. Orientation to CS Issues
Student hours: 10
Personnel Categories: All
2. Center Organization/Mission
Student hours: 1
Personnel Categories: All
3. Fundamental Beliefs
Student hours: 2.5
Personnel Categories: I, II, III, IV, V, VI, VII
4. Policy, Dir, Regs, Legalities
Student hours: 3
Personnel Categories: II, V, VI, VII(1)
5. Classification
Student hours: 1
Personnel Categories: I(2), II(2), III(2), IV(2), V(2), VI(2), VII(2), VIII(2), IX(2)
6. Ethics/Responsibility
Student hours: 4
Personnel Categories: All
7. Measuring Computer Security
Student hours: 1.5
Personnel Categories: I, II, III, IV, V, VI, VII, VIII
8. Criteria II
Student hours: 3
Personnel Categories: I, II, III, IV, V, VI, VII
9. Evaluating the Environment
Student hours: 1
Personnel Categories: II, V, VI, VII(1)
10. Risk Management
Student hours: 1
Personnel Categories: II, V, VI, VII(1)
11. Administration of Computer

Science

Student hours: 1
Personnel Categories: II, VI

12. COMSEC Overview
Student hours: 2
Personnel Categories: I, II, III, IV, V(1), VI, VII, VIII(1), IX(1)
13. Architectures
Student hours: 20
Personnel Categories: I, II, III, IV, V(1), VII(1)
14. The Criteria (tech version)
Student hours: 20
Personnel Categories: I, II, III, IV, V(1), VII(1)
15. Theoretical Foundations
Student hours: 60
Personnel Categories: I, II, III, IV, V(1), VII(1)
16. Model Interpretation
Student hours: 16
Personnel Categories: I, II, III, IV, V(1), VII(1)
17. Correctness
Student hours: 60
Personnel Categories: I, II, III, IV, V(1), VII(1)
18. Evaluation Theory and Practices
Student hours: 40
Personnel Categories: I, II, III, IV, V(1)

(1) job-specific; may be required
(2) required only for new hirees or others with insufficient experience in dealing with classified materials

TOTALS:

I=13, II=17, III=13, IV=13, V=9, VI=11, VII=7, VIII=4, IX=3

PERSONNEL CATEGORIES:

I = product evaluator
II = system evaluator
III = R&D specialist
IV = technical implementation specialist (BLACKER)
V = manager
VI = trainer
VII = support
VIII = administrator
IX = clerical

DESCRIPTION OF TRAINING
MODULUES
(Enclosure 4)

TOPIC: 1. Orientation to
Computer Security Issues

TIME FOR STUDENT TO COMPLETE:
10 hours

SUMMARY OF MODULE: Basic
Theme: What is really going on
when a computer works; break
the "hallucination" syndrome

1. Standard Terminology and
Basic Concepts

A. How a Computer Works

B. Computer Subversions

Trojan Horse
Trap Door
Time Bomb (Logic Bomb)
Data Diddling
Salami Technique
Superzapping
Virus

The results of these
subversions:

Destruction (Denial of
Service)
Alteration of Data
Disclosure of Data
Delay (Down Time)

C. Definitions

Access Control
Controlled Sharing (not in
COMPUSECese)
Reference Monitor
Security Kernel
Trusted Computing Base
System High Operations
Dedicated Operations
Controlled Operations
Multilevel Operations

2. Lines of Defense

A. Physical

Various devices to prevent
theft, damage, or destruction
to a computer facility or its
components.

List devices
Give major problems and
examples

B. Personnel

Measures taken by
management to ensure that
employees in ADP-related
positions are both

knowledgeable and trustworthy
in matters of computer
security.

List measures
Give major problems and
examples

C. Communications

The means of ensuring that
information passing through
communications channels is
protected from unauthorized
access and interpretation.

Describe areas of concern;
Explain method of protection
(cryptography)

D. Emanations

Way of ensuring that our
electronic equipment does not
radiate signals that can be
collected by an adversary.

Describe problem
Explain method of protection

E. Operational Procedures

Policies and rules that
ensure that actual practices in
the computer facility or area
adhere to principles of
security.

Automated audit and
individual accountability

List recommended procedures

Describe operational
environments using secure
procedures.

F. Trusted Computer Systems
(TCS)

Components of a TCS -
namely, hardware, software, and
configuration control - provide
enough protection to ensure
that a range of classified and
sensitive information can be
processed simultaneously
without danger of compromise.

Define hardware, software,
and configuration control.
Describe briefly how these
areas can be protected or are
evaluated.

3. Threat and Vulnerability

A. Threat - external and
internal

B. Vulnerability

(1) Mainframe Vulnerabilities

The vulnerabilities we are most concerned about, those that may occur quite frequently. Most of these frequently occurring vulnerabilities are present because security was not a design issue. We can group these recurrent vulnerabilities into three categories. The first category is the improper use of technology; this category includes: insufficiently trained operators, poor applications, data entry errors, and improperly designed multiuser connections. The second category encompasses vulnerabilities generated by weak or non-secure operating systems. These include trap doors left by system developers, easily gained super-user (super-zapper) status, and microcode/assembly language manipulation of operating system controls. The third category of vulnerabilities are improper access controls such as poor log-on procedures, weak password management, and trivial audit procedures. Through the use of a trusted computer system many of these vulnerabilities can be alleviated.

(2) Personal Computers

- Hardware Security Concerns
 - A. Theft and Damage
 - B. Equipment Aids
 - C. Environmental Controls
 - D. Magnetic Media
- Information Security Concerns
 - A. Theft and Damage of Data
 - B. Contamination of Data
- Software Security Concerns
 - A. Piracy
 - B. Risks of borrowed software: viruses and integrity issues
- Communications Concerns

TOPIC: 2. Center Organization and Mission

TIME FOR STUDENT TO COMPLETE:
1 hour

SUMMARY OF MODULE:

This module will contain a

review of the Center organization to division-level and a brief description of the activities in each entity. The discussion will also show how the Center fits within NSA, DoD, and the Intelligence Community. Our special national mission will also be explained, and how the Center carries out this mission through the NTISSIC structure will be addressed.

(Must be taken before Module #3)

TOPIC: 3. Our Fundamental Beliefs and Policy...The Catechism

TIME FOR STUDENT TO COMPLETE:
2.5 hours

SUMMARY OF MODULE:

The employee will first read the Catechism and then participate in a discussion of these issues, which will be led by a senior Center policy maker.

The Catechism discussion forum will be held approximately once a quarter.

(Module #2 is pre-requisite)

TOPIC: 4. Policy, Directives, Regulations, and Legalities

TIME FOR STUDENT TO COMPLETE:
3 hours

SUMMARY OF MODULE:

An hour lecture will highlight the significant features of those directives, regulations, and other documents which govern security in the Federal Government. Appropriate DoD and OMB policy and implementation documents will be examined in detail. After the lecture, copies of the referenced documents will be made available for the students' study, with guidance from the supervisor on which documents are most germane to the students' work.

TOPIC: 5. Fundamentals of Classification

TIME FOR STUDENT TO COMPLETE:
1 hour

SUMMARY OF MODULE:

The briefer will review the fundamentals of the DoD classification system. Specifically, the briefing will include the NSA Act of 1959 and a review of Public Law 86-36. The four types of protected information, need-to-know, and the classification categories will be discussed. Covernames and codewords will be defined and the reason for them will be explained.

TOPIC: 6. Ethics and Responsibilities of Center Personnel

TIME FOR STUDENT TO COMPLETE:

4 hours
(2 hours of reading, 1 hour of videotapes, 1 hour of discussion)

SUMMARY OF MODULE:

I. Computer Usage (in general and as an individual)

A. Responsibility Defined - legal and security requirements

B. Issues In a Computer Information Society

Unauthorized access
Ownership of Information
Giving one's password to an unauthorized user
Privacy
Copywrite violation or piracy

C. The Center as a Showcase

II. Government Employees' Responsibilities

A. NSA, DoD, and Government standards of conduct and related rules and regulations

B. Our Relationship with Non-Government Organizations

1. DoD Community Relations Program

- a. Objectives
- b. Policies

2. Participation in commercially sponsored conferences/symposia

3. Participation in activities of private organizations

4. Providing information to

non-Government organizations

5. Dealing with contractors

- gifts
- unfair advantage

C. Handling of Sensitive (Unclassified) Information

1. Sensitive Information Defined

a. Unclassified info which may be protected by P.L. 86-36

b. Information protected by PL 93-579, the Privacy Act

2. Responsibilities for protecting sensitive information

- a. Physical Protection
- b. Need to know
- c. Privacy Act Restrictions
 - 1. The Law
 - 2. Internal Rules

D. Handling Proprietary Information

III. The Media; Publication Procedures

A. Release of Unclassified Information

1. Written information:

- a. presentations
- b. school Papers
- c. books
- d. personal records
- e. logos
- f. business cards

2. Central point of control necessary to:

a. Oversee cumulative effect of information leaving Agency

b. Coordinate with SECDEF, DoD, and others, as appropriate

c. Serve best interests of Agency

3. Procedures for requesting release

B. Media inquiries

1. Central point of control necessary as in (A) above

2. Procedures for handling

- a. Verbal inquiries
 - 1. telephone inquiries
 - 2. inquiries received during conferences, symposia,

etc...

b. Written inquiries
IV. Behavior of a Center Representative

A. Code of Ethics for Government Service

1. Matters of ethical conduct include:

- a. Business and Professional Activities
 - b. Bribery and Graft
 - c. Gratuities
 - d. Contributions or Presents to Superiors
 - e. Use of Government Facilities, Property and Manpower
 - f. Use of Civilian and Military Titles in Connection with Commercial Enterprises
 - g. Outside Employment
 - h. Gambling, Betting and Lotteries
 - i. Personal Indebtedness
2. Responsibilities of Employees
3. Responsibilities of Managers

B. Conflicts of Interest

1. Major Prohibitions
2. Non-Disqualifying Financial Interest
3. Procedural Requirements

TOPIC: 7. Measuring Computer Security

TIME FOR STUDENT TO COMPLETE:
1 1/2 hours

SUMMARY OF MODULE:

The purpose of this module is to acquaint new employees with the purpose and thrust of the Criteria, how it fits into the Center's mission, and its utility as an instrument of policy. This module is basically intended for non-technical staff, and thus will not delve into the details of Criteria requirements. Although some of the material will parallel that given in the Criteria module for technical personnel, it will generally be presented in considerably less depth. The student should be

left with an understanding of what the Criteria is about, its uses, and its importance to the Center and Center policy and technical directions.

a. Develop the Need for a Criteria: This section is designed to lead the student to an appreciation of the experiences, problems, and solutions that led to the attempt to write the Criteria, and thus lead the student to an appreciation of the value of the Criteria. Basically, the discussion should proceed as follows:

- DoD experience with "custom-built" systems; the problems of non-common terminology, non-common perception and articulation of requirements, all of which leads to expensive systems which still may not provide the level of security desired.

- Utility/purpose of the Criteria; provide common understanding of the fundamental security issues, provide a common terminology, and provide a common yardstick for measuring and comparing security "goodness."

b. Derivation From Policy: This section is designed to lead the student to an understanding of the issues the Criteria addresses, and why it addresses those issues. It should be approached from the direction of "Let's design computer security criteria." Selections from relevant national policy should be presented. This should be in "plain English" as opposed to DoD jargon. The idea is to demonstrate the need to derive requirements from basic policy statements, and to develop the Criteria control objectives. The student will be introduced to the basic concepts underlying the Criteria, such as policy enforcement (to include DAC and MAC), individual accountability, and auditing. The student should be led to an appreciation of the place of each control objective in overall security fabric.

c. Structure of the Criteria: The purpose of this section is to develop a familiarity with the D through

All terminology, where it comes from, and what it means. The structure of the Criteria will be presented as a scale for measuring, with essentially no details. Major distinctions between the Criteria divisions will be characterized.

d. Uses of the Criteria:

Re-emphasize the two basic uses to which the Criteria are put, basically;

- As a tool for determining system security requirements.

- As a yardstick for measuring security "goodness" of products.

Each of these should be discussed in the context of the Center missions they support.

e. The Criteria in Detail:

In this section the student will be taken on a detailed tour of each Criteria division and class. Each division will be characterized, and then the specific requirements of each class will be discussed. Because this section is specifically intended for those who require extensive knowledge and deeper appreciation of the Criteria, observations relevant to implementation choices/difficulties are appropriate. The results of Criteria interpretation, as well as any insights gained from the interpretation process (i.e., difficulty of applying the Criteria to some situations, e.g., VMM) will also be presented. The "breakpoints" (i.e., B1/B2, B2/B3) will be studied. Also, evaluation issues and implications will be incorporated into the presentation (e.g., what is sufficient evidence to support the requirements for formal specification and verification?).

TOPIC: 8. Criteria Part II - The Requirements of Each Class

TIME FOR STUDENT TO COMPLETE:
3 hours

SUMMARY OF MODULE:

This sub-module is designed for the person who needs or desires more detail on the technical content of the Criteria. It is expected that

this material will be presented primarily to educators and managers. The material is a prerequisite for this sub-module. The student should be left with an understanding of how the Criteria move from an emphasis on features to an emphasis on assurance, some understanding of the details of the various classes and divisions of the Criteria, and an appreciation of how the Criteria also serve as an instrument of Center policy.

a. Structure of the Criteria: Should start with a somewhat more thorough tracing of the Criteria from rationale, basic principles, and the reference monitor concept, all of which are derived from basic policy statements. Here it is desirable to show which documents state which requirements. Some of the basic concepts can be expanded upon, notably MAC and DAC mechanisms, getting into more detail as to the policy requirement and implementation implications. Additionally, other concepts, such as Trojan horse and covert channels, can be introduced. The details of the Criteria will be presented as follows:

- C1 through B1 will be addressed as a group, noting that the architectural/assurance requirements are similar across these classes. Discussion of each of these classes in detail, showing the progress from one class to the next higher one. Distinguish between mechanisms (e.g., DAC) and assurance items (e.g., object reuse). Note that the emphasis is largely upon the addition of mechanism, thus the Center view that it is possible to "grow" a B1 system from a D system merely by adding features.

- B2 Systems: will be presented as a separate subject, noting the distinguishing characteristics of this class. The student should be left with an understanding of the requirements for basic system structure and architectural support for security that underlie this class. The student should understand these aspects of the requirements as the beginnings of real

assurance, noting that at this juncture in the Criteria the emphasis changes from adding mechanism to adding the assurance. B2 is the first level in which we are assured that a reference monitor function is credibly implemented.

- B3 and A1; also to be presented as a unit, noting that A1 is architecturally equivalent to B3 (i.e., no new features are added). The strengthening of the top-down design requirements and demand for more thorough architectural support should be explored, and examples from the Criteria presented.

b. Relation to Policy and Strategy: Here we state the Center position that we will always specify ADP system security requirements in terms of Criteria ratings (vice the "Chinese menu" approach). Discuss how this is consistent with, and in fact follows from, the Center mission to make improved products widely available in the marketplace. Discuss the "chicken and egg dilemma," and what is being done to address it (e.g., the EPL, influencing the RFP process, national-level policy, environments document, etc). Discuss the export control issues, noting the Center position as well as the current status of the U.S. export control policy. Explore what steps the Center is taking to continue to encourage vendors to cooperate. Can also discuss here how the Criteria is a tool in determining the R&D directions. The basic thrust of this section is to leave the student with an understanding of the Criteria as a document which, derived from basic policy statements, articulates fundamental security requirements. Thus it also serves as an instrument of Center policy and a guiding tool for charting future directions.

TOPIC: 9. Evaluating the Environment

TIME FOR STUDENT TO COMPLETE:
1 hour

SUMMARY OF MODULE:

The rationale behind the

Center Environments document will be discussed, especially the development of the Risk Index. Then some real-world examples of how to apply the recommendations in the document will be discussed.

TOPIC: 10. Risk Management

TIME FOR STUDENT TO COMPLETE:
1 hour

SUMMARY OF MODULE:

Risk management is the identification of risks to an organization's information resources through an analysis of information assets, threats, and vulnerabilities. Key terms which must be understood are:

asset
threat
vulnerability
risk
loss
safeguard

The module will cover the purpose of risk management and the methods involved in conducting a risk analysis project. An example will be presented which will illustrate this process.

TOPIC: 11. Administration of a Computer Security Program in an Organization

TIME FOR STUDENT TO COMPLETE:
1 hour

SUMMARY OF MODULE:

This module describes security program management considerations. It consists of basic guidelines for establishing and managing a computer security program. Specifically, these topics are covered:

1. Elements of a good computer security program
2. Pitfalls to Avoid for Managers
3. Senior Management Duties in a Computer Security Program
4. Internal Control Considerations for Managers
5. Audit Function Considerations
6. Making Deliberate Business Decisions
7. Balancing Technology

and Human Issues
8. Setting and
Implementing Goals - Managerial
Considerations
9. Managing Computer
Employees
10. Making Computer
Security Work

TOPIC: 12. COMSEC Overview

TIME FOR STUDENT TO COMPLETE:
2 hours

SUMMARY OF MODULE:

This topic is meant to give the student an appreciation of the COMSEC threat and some countermeasures that can be used. Some basic concepts of encryption, including the DES, will be covered.

A variety of videos and readings will be available. The exact videos to be viewed will be determined by the employee's supervisor, selecting those most germane to the employee's job.

For example, clerical personnel may benefit most from material emphasizing the vulnerabilities of telephones and other office systems.

TOPIC: 13. Architectures

TIME FOR STUDENT TO COMPLETE:
20 hours

SUMMARY OF MODULE:

The purpose of this module is to provide the student with an understanding of the major computing architectures, and especially how protection mechanisms are incorporated into hardware and software systems. It will provide the basis upon which to build an understanding of the architectural and design implications of the Criteria, and to explore how specific architectures (e.g., stack, descriptor, capabilities) support (or do not support) the Criteria. Liberal use should be made of case studies; the idea is to use real systems to illustrate the points under consideration.

a. Basics: This section will introduce the student to fundamental concepts,

terminology, and mechanisms. Various architectures will be described, such as descriptor-based, stack, and object-oriented systems. Additionally, memory management and process management strategies will be explored.

b. Models of Protection Mechanisms: Lampson's Access Matrix model will be presented. The notion of domains, objects, access privileges, and rules for their manipulation will be presented as examples of operational models of the Access Matrix model. The student should be left with an understanding of the issues of domain isolation, authorization of domain access to objects, the transfer, revocation, and review of access privileges between domains, as well as the creation and destruction rules for both domains and objects.

c. Architectural Support for Domains of Protection: Various interpretations of the domain model are considered, which lead to descriptor and ring-based protection mechanisms, capability-based systems, storage-key and privileged-mode protection mechanisms, domain call-return mechanisms, and stack frame protection. Each of these will be related to the issues identified above (i.e., domain isolation, etc.)

d. Implementation of Protection Mechanisms: Here we discuss the implementation issues of protection mechanisms. The relationship between protection mechanisms and the addressing and virtual memory mechanisms will be discussed. The impact of various implementation choices (e.g., multiprocessors, pipelining, caches, address translation buffers, and I/O architectures) will be examined. Explore trade-offs between hardware and software implementation.

e. Case Studies: Specific architectures are studied in the light of protection requirements and, specifically, the above material. The pros and cons of each architecture are discussed. Performance aspects

may be brought in here.
Candidate architectures to be studied include:

- MULTICS
- SCOMP
- INTEL 432
- BURROUGHS 5500
- IBM 370

TOPIC: 14. The Criteria
(Technical Version)

TIME FOR STUDENT TO COMPLETE:
20 hours

SUMMARY OF MODULE:

The purpose of this module is to acquaint new employees in technical fields with the purpose, thrust, and structure of the Criteria, and to present the justification for its essential elements and characteristics. Although the material will go into considerable depth on the details of the Criteria requirements, it is not intended as "the last word" on Criteria tutorials; it will leave sufficient room for further study into the Criteria itself, as well as related areas. The student should be left with an understanding of the scope of the Criteria, the fundamental issues with which it deals, the ways in which it deals with them, and the utility of the Criteria. The module should provide the basis for studying the Criteria in greater depth, and in fact will provide the student the base upon which the final module ("Evaluation Theory and Practice") builds to develop a much deeper appreciation of the implications of the various criteria requirements. This module will also discuss the role of the Criteria as an instrument of Center policy.

a. Purpose of the Criteria: This section should set the stage by demonstrating the need for a common knowledge base from which requirements can be stated in a consistent manner. The primary purposes of the Criteria to be discussed will be:

- articulate fundamental issues, requirements. It should be noted here that the Criteria is presented in terms of requirements, and does not

mandate implementation; it purposely leaves room for the vendor to make implementation choices.

- provide the basis for an objective and consistent metric of "security goodness."

This section will also discuss the difference between internal controls and external controls (e.g., procedures, physical security, personnel security), making it clear to the student that the Criteria focuses only upon internal control mechanisms.

b. Derivation From Policy: Discuss the philosophical underpinnings of the Criteria. Show that it is not merely an arbitrary collection of good ideas but rather it is derived from basic national policy requirements and well understood security and scientific principles. Show how the "control objectives" are derived. At this point several fundamental concepts will be introduced and studied in some detail, such as MAC and the lattice model, DAC (need-to-know mechanisms), individual accountability, and labeling.

c. Structure of the Criteria: Start off with the basic elements of the Criteria, tying them back to the control objectives, and distinguish between features/mechanism and assurance elements. The structure should be presented in overview (i.e., D to A) first. This will give a global perspective before delving into detail, and allow a chance for presenting the justification for choosing a linear (vice multi-dimension) rating scheme. Next, each Division and Class will be studied in some more detail, touching only upon the main elements of each division and class. What is important here is to discuss the essential characteristics of each class, and to show how the Criteria progresses from an emphasis on mechanism, at the lower levels, to an emphasis on assurance at the higher levels.

- Optional - discuss the question of "beyond A1"; prognostications. Can be used to show how the basic technological and policy

thrusts of the Criteria can be extended, as well as to show the Criteria is limited by the state-of-technology and at the same time provides the base from which technical direction can be mapped.

d. **Related Topics:** Prior to getting into the fine-grain detail of each class of the Criteria, some attention should be given to aspects of the Criteria that are not apparent from a superficial reading. In particular, the topics to be covered will include:

- What is not included and why; denial of service, reliability, and integrity

- Rating scale; examine the choices of one-dimensional rating vs. a multi-dimensional rating. What considerations led us to the choice we made.

- Relation to policy/strategy. Here we will note the Center position that we will always specify ADP system requirements in terms of Criteria ratings (vice using the Criteria in a "cut-and-paste" mode). This position will be shown to be consistent with the Center's mission to make improved products available in the marketplace. It should also be shown to be supportable on the technical grounds, that each Criteria class is essentially defined by its characteristic assurance elements (i.e., a B2 is a B2 regardless of how much chrome trim is added or left off).

TOPIC: 15. Theoretical Foundations

TIME FOR STUDENT TO COMPLETE:
60 hours (2 hours daily for 6 weeks)

SUMMARY OF MODULE:

1. In any security evaluation it seems reasonable to begin by establishing the security policy which guided the designers of the entity. New technical colleagues need to be acquainted with the concept of the security policy followed by the various methods by which it has been expressed. This leads, naturally, to the notion of the

model as the tool for describing in precise language the elements of the security policy. In order to make a meaning presentation of the models it is necessary that the students have a minimal set of logical and mathematical notions at hand. For example, they need to know a few notions in set theory and modern algebra (partially ordered sets and lattices.) Consequently, a minimal presentation would proceed along the following lines:

a. A heuristic exposition on the notion of security policy with emphasis on defining security in the context of a computer system.

b. Basic notion of a model as a device for defining precisely informally expressed concepts.

c. Basic mathematical concepts needed to understand existing models for secure computer systems. (If one objects to the use of the word "mathematical," it can be replaced with "logical.")

d. Basic notions of access control. A good survey is found in Chapter 4 of Denning's book. (It is not intended that the entire chapter be covered.)

2. When topics a., b., c., and d. above have been covered, the students are ready to look at the models themselves. One would then proceed to present:

- a. Bell LaPadula Model
- b. Information Flow Model
- c. Non-Interference Model
- d. Multi-level objects (NRL MMS model; SYTEK model)

3. It would be helpful, if time permits, to give examples of existing systems, or those in the design process, which incorporate the models described in 2., or variations thereof.

TOPIC: 16. Model Interpretations

TIME FOR STUDENT TO COMPLETE:
16 hours

SUMMARY OF MODULE:

The purpose of this module is to demonstrate how the formal policy model and the

reference monitor concept is actually embodied in lower levels of abstraction (i.e., implementation detail). The main thrust will be to show how step-wise decomposition (i.e., top-down design, development) of the design provides the basis for the convincing argument that the ultimate hardware/software system in fact enforces the rules of the policy model, and also provides the necessary reference monitor qualities (i.e., self-protecting, always necessary to be invoked, small enough to be analyzed). The detailed module would proceed as follows:

a. Interpretation of the model: Note the intellectual gap between the different levels of abstraction represented by the formal policy model and the FTLS (which begins to include significant implementation detail). Show how that intellectual gap is addressed through the arguments that map the state transition rules of the formal policy model to functions of the particular architecture (e.g., descriptor-based, stack) to be implemented. Case studies which can be used to support this section include the "Unified Exposition and Multics Interpretation" (Bell and LaPadula), Scomp Interpretation, and Multics Interpretation (Multics B2 evaluation).

b. Formal Top-Level Specifications (FTLS): Introduce the basic principles of formal, top-level specifications and proof-of-correctness (or verification) techniques. Discuss the correct level of abstraction of the FTLS, principles of FTLS design, what defines the user/TCB interface. Introduce the concept of "trusted subjects"; what constitutes "trustedness," what is the role of a trusted subject (i.e., why is this construct needed?) Explore the relation of the FTLS to the Reference Monitor Concept; which aspects of a reference monitor are addressed by the FTLS, which are not. Explore the distinction between functional correctness and proper security behavior, and

that the verification technology addresses the latter issue.

Optional - Discuss covert channels; what are the issues, where do they come from, what are the design considerations if they are to be eliminated?

c. Spec-to-Code Mapping: How the FTLS are carried into lower levels of implementation detail and, eventually, into source and object code. Presented as the continuation of the argument that the rules of the formal policy model are enforced at each level of design, as more and more detail is introduced. Discuss this set of steps as a consequence of the limits of the state-of-technology in verification (i.e., will not be necessary at such time as verification of source/object code is a reality). Demonstrate the necessity for showing that the following conditions are both true:

- all the code that appears in the TCB is directly derivable from the FTLS; no additional functionality is introduced. All that is added is implementation detail.

- implementation detail which is not described at the FTLS level represents only non-user-visible functionality; represents detail which is not at the TCB interface.

Useful case study for this section is the SCOMP spec-to-code mapping, preceded by a reading of the MITRE paper on this subject (Benzel).

TOPIC: 17. Correctness

TIME FOR STUDENT TO COMPLETE:
60 hours (2 hours daily for 6 weeks)

SUMMARY OF MODULE:

- I. INTRODUCTION
 - A. Background Knowledge
 1. Formal logic
 2. Set theory
 3. Modeling
 - a. notion of using precise language
 - b. exposure to expressing abstract concepts formally
 - B. Basics in a Nutshell

1. What is formal specification?
2. What is verification?
3. What good is it? Why use it?
4. Role of verification in developing secure systems.

II. THEORY

A. Read and Discuss Seminal Papers

1. Floyd's "Assigning Meaning to Programs"
2. Hoare's "An Axiomatic Basis for Computer Programming"
3. Hoare's "Procedure and Parameters: An Axiomatic Approach"

B. Special Topics

1. Information flow tools
 - a. recommend some of John Rushby's papers out of SRI explaining the theory and implementation of the tools.
 - b. advantages
 - c. restriction & limitations of the approach

III. THE REAL WORLD

A. Gypsy as a Reflection of Floyd/Hoare Theory

1. Brief description of Gypsy
 2. How it's used
 - a. read and discuss "Model and Design Proofs in Gypsy: An Example Using Bell and LaPadula."
 - b. possibly read and discuss section of "Using the Gypsy Methodology." It depends on the time available.

B. Applications

1. The EPI (Encrypted Packet Interface) work done at Texas. Recommend reading "Formal Verification of a Communications Processor"
2. Possibly read and discuss the paper in the Scientific Honeyweller of July 1985 entitled "Proving a Computer System Secure."

C. How to Evaluate

1. Analyze and understand the approach taken. Read and discuss "Structuring a System for AI Certification." Also read and discuss Platek's paper on problems with Feiertag tool and HDM to see the dangers of placing too much confidence in a set of tools simply because they are on a computer.
 2. Ask "What's being proved?"
 3. Ask "What's being

assumed?"

TOPIC: 18. Evaluation Theory and Practice: Putting the Criteria to Use:

TIME FOR STUDENT TO COMPLETE:
40 hours

SUMMARY OF MODULE:

The purpose of this module is to give the student a full appreciation of the Criteria and its implications. It will give the student both vocabulary and true understanding of the scientific principles underlying the Criteria, which will allow him to be able to present/discuss the Criteria from a firm technical base. It will use the Criteria as a central focus in order to consolidate all the preceding technical material. The approach will be to study each major class of products (i.e., C1 - B1, B2, B3 - A1) with a view to how the concepts of modeling and assurance are embodied at each level, and what the architectural implications are on each level. The presentation should be in the context of choosing logical building blocks for converting high-level models of access control and policy into working systems which enforce the necessary constraints; how to progress from abstract design to end product in such a way that convincing arguments can be made for the correct security behavior of the end product.

a. C1 Through B1: The approach here, as it will be for each of these sections is basically: "what is required to build one; what does it mean to satisfy the requirements?" The issues to be discussed will be:

What are the architecture issues? Basically, what are "credible controls capable of enforcing access limitations..." (C1); and what are the architectural implications? What are the implications of the requirement that the "TCB...maintain a domain for its own execution that protects it...?"

What are the assurance issues? What counts for a "security policy model" at

these levels of the Criteria, and what are the "convincing arguments" which can be made for believing that the resultant system in fact provides the level of protection desired?

What are the implementation choices; what tradeoffs can be made? What specific architectures provide the qualities desired?

b. B2: Primarily the same discussions as above. A major discussion should revolve around the question "how are the B2 requirements, and thus the resulting architecture fundamentally different from an architecture which satisfies the C1-B1 requirements; why, and how, is a B1 architecture not adequate for B2?"

c. B3 to A1: Same approach as above (details to be worked out by course designer).

d. Lab: Perform a sample evaluation; students will show reasoning used to decide what level of Criteria is satisfied by the system being studied. Teams of 2 or 3 students will evaluate an appropriate device (real or imaginary) and produce a written report.

TRAINING ROAD MAP

BASIC ISSUES:

```

:-----:
:  Orientation  :
:    to         :
: Computer Security :
:    Issues     :
:-----:
  
```

```

:-----:
: Fundamentals of :
: the Catechism  :
:-----:
  
```

```

:-----:
: Center Organization:
: and Mission       :
:-----:
  
```

```

:-----:
: Policy, Directives, :
: Regulations, and    :
: Legalities          :
:-----:
  
```

```

:-----:
: Classification     :
:-----:
  
```

```

:-----:
: Ethics and Responsibility:
:-----:
  
```

```

:-----:
: Potpourri         :
:-----:
  
```

```

:-----:
: Measuring Computer :
: Security            :
:-----:
  
```

TECHNICAL ISSUES:

```

:-----:
: Theoretical       :
: Foundations       :
:-----:
  
```

```

:-----:
: Model            :
: Interpretation   :
:-----:
  
```

```

:-----:
: Correctness     :
:-----:
  
```

```

:-----:
: Architectures   :
:-----:
  
```

```

:-----:
: The             :
: Criteria        :
:-----:
  
```

```

*****
*                                     *
*                                     *
*                                     *
* ----- *
* :Elements of a :
* :C1-B1 System  :
* :----- *
* :----- *
* :Elements of a :
* :B2 System     : A
* :----- *
* ----- Potpourri *
* :----- *
* :Elements of a :
* :B3-A1 System  :
* :----- *
*****
  
```



DEPARTMENT OF THE NAVY AUTOMATED DATA PROCESSING
SECURITY PROGRAM TRAINING
OPNAVINST 5239.1A

Patricia Grandy

Navy Regional Data Automation Center San Francisco
NAS Alameda, CA 94501-5007
COMM: (415) 869-5300
AVN: 686-5300

ABSTRACT

The Naval Data Automation Command (NAVDAC) is an echelon II command of the Chief of Naval Operations. It consists of a headquarters staff located in Washington, D.C. having echelon III and IV Automated Data Processing (ADP) support activities known as Navy Regional Data Automation Centers (NARDACs) and Navy Regional Data Automation Facilities (NAVDAFs). NAVDAC activities are found in most regions of the United States where there is extensive Navy activity.

The Commander, Naval Data Automation Command (COMNAVDAC) conducts training for all Department of the Navy (DON) and Marine Corps activities (Shore and Afloat) and DON contractors. The DON ADP Security Program is established by OPNAVINST 5239.1A, an instruction which consolidates all pertinent ADP security information on policies, procedures, and responsibilities for establishing and maintaining ADP security programs at all levels within the DON.

In implementing an activity ADP security program, one of the biggest obstacles facing the Commanding Officer is developing a command awareness of ADP security. The DON approach to a problem of such magnitude as ADP security, is to analyze the problem and find solutions through Risk Assessments. A four day "Introduction to the DON ADP Program" Course provides an awareness of the ADP security problem and the need for a Navy ADP Security Program. The course attendees are middle management (GS-9 and above, E-7 and above) assigned as ADP Security Officers, ADP System Security Officers, Network Security Officers, Terminal Area Security Officers or others with an interest in ADP security. Class size is maximum thirty-six and quotas are limited to two attendees from a command to ensure an equitable distribution of experience. The course schedule combines lecture, outside reading, and workshops with a modular workbook covering twenty-five areas. The course includes a challenging case study to present the two DON Risk Assessment Methodologies. Method I instruction involves conducting workshops to systematically study assets, their weaknesses and strengths, and possible threats; determining the probability of a successful attack occurring and the dollar value of its impact; and conducting a cost/benefit analysis of implementing additional countermeasures to achieve an optimum level of security. Method II is an abbreviated methodology for Risk Assessment.

This approach is suitable for less complex configurations and/or microcomputers.

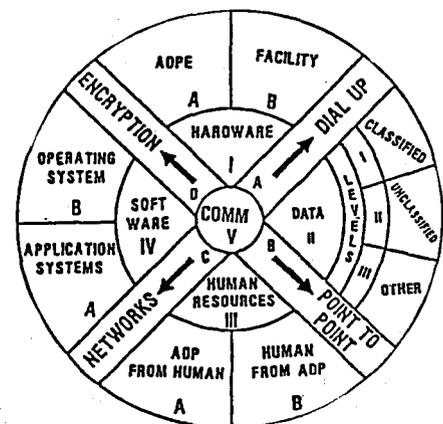
NAVDAC, through the NARDACs and NAVDAFs located in San Francisco, Washington, D.C., Jacksonville, Newport, and Pearl Harbor, conducts more than fifty ADP Security classes annually, at locations all around the world.

DON ADP SECURITY PROGRAM TRAINING
OPNAVINST 5239.1A

The purpose of the DON ADP Security Program Training is to provide ADP Security Staff personnel with an overview of the Navy ADP Program, which includes defining the scope of the Navy ADP Program, providing an awareness of the ADP security problem, and emphasizing the need for a working activity ADP Security Program.

OPNAVINST 5239.1A, the ADP Security Manual, is a directive. A directive requires compliance. What is "ADP Security" all about? In a few words, it is the means for protecting our investment in automated data processing. In our ADP "portfolio", we have invested many dollars and much time in the five asset areas defined in the DON ADP Security Program. They are:

- I. HARDWARE
- II. DATA
- III. HUMAN RESOURCES
- IV. SOFTWARE
- V. COMMUNICATIONS



DON ADP SECURITY (FIVE AREAS)

The DON ADP Security Course consists of the following modules presented in a combination of lecture, conference, and workshop sessions.

DON ADP SECURITY POLICY

The objective of this session is to inform the students that the current Navy ADP Security Policy is a composite of existing Department of Defense (DOD) and Navy ADP security requirements. The discussion includes each specific element of DOD and Navy policy upon which the Navy's ADP Security Program is presently based.

ADP SECURITY PROGRAM RESPONSIBILITIES

This session helps the students to understand the distribution of policy-making, program management, operating and program review responsibilities of National Agencies, and Navy Offices involved in the DON ADP Security Program. The students are instructed on the individual responsibilities of the Designated Approving Authority (DAA), the Commanding Officer and the ADP Security Staff members and will be able to explain how these individuals interact to support an ADP Security Program at the Navy activity level. The DON ADP Security Staff consists of an ADP Security (ADPSO), a Network Security Officer (NSO), ADP System Security Officers (ADPSSO), Terminal Area Security Officers, an Office Information System Officer, the activity Security Manager and Security Officer, as required. Additional security personnel, such as Top Secret Control Officer and CMS Custodian, are discussed as they interface with the ADP Security Staff.

ACCREDITATION OF NAVY ADP ACTIVITIES AND NETWORKS

The objective of this session is to instruct the student on the accreditation concept and provide guidance on how to apply these concepts to their own activity. A Statement of Accreditation is the DAA's formal declaration that an appropriate security program has been implemented for an activity's systems or networks consistent with Levels I, II, and III data protection requirements.

SECURITY OF NAVY OFFICE INFORMATION SYSTEMS (OIS)

The objective of this session is to apply the knowledge of DON data levels to determine how OIS are to be secured and which accreditation elements apply to the security of these systems in their own activities.

MICROCOMPUTER SECURITY

This session discusses security requirements for Personal Computers (PCs), as well as, suggestions for developing an activity policy on (1) privately-owned PCs accessing DON data from non-government controlled workspaces (e.g., home); (2) the use of privately-owned software and data for government business; and (3) privately-owned PCs, software and data brought into government controlled workspaces. This session emphasizes developing an activity policy on adherence to software licensing agreements for copyrighted software packages.

AFLOAT SECURITY

Shipboard computer systems provide unique considerations with respect to ADP Security. This session discusses physical security requirements for afloat units when underway, in foreign ports, and in drydock or a shipyard, TEMPEST certification (policy and guidance from Type Commanders), and shipboard ADP Security certification for particular computer systems, such as, the Shipboard Non-Tactical ADP Program II (SNAP II).

THE PRIVACY ACT

This session is designed to make the students aware of the significance and impact of Public Law 93-579 (The Privacy Act of 1974). The students are instructed in how to apply Conditions for Disclosure of Information, identify releasable information, and explain agency requirements.

MINIMUM ADP SECURITY REQUIREMENTS

This session teaches the students to relate Navy ADP security requirements to their own activities, recognizing any deficiencies in existing ADP security programs. Minimum mandatory requirements include environmental controls (temperature, humidity, lighting, electrical power, cleanliness, water damage, fire safety, smoke detection, etc.), physical security (facility, remote terminal areas, disconnect procedures, control zones, etc.), communications security, emanations security, hardware/software security, and contingency planning.

EMANATIONS SECURITY

Emanations security discusses measures to control compromising emanations (EMSEC), which are required under the provisions of DOD S-5200.19, Control of Compromising Emanations (U) and supplemented by OPNAVINST C5510.93D. Students are made aware of the risks associated with using equipment which produces compromising emanations, to enable them to recognize the various countermeasures to be implemented at their ADP facility. The session discusses how to initiate requests for TEMPEST Vulnerability Assessments (TVARs) for all ADP and OIS systems for processing Level I data and the necessary procedures to follow to obtain TEMPEST Accreditation.

SECURITY OF ADP MEDIA

This session is designed to enable the students to apply the requirements of both the Navy information and ADP security programs to marking, accounting for, and handling Level I (classified) and Level II data recorded on ADP media.

ADP SECURITY SURVEYS AND CHECKLISTS

This session provides the students an instruction in the use of a standard ADP security survey format to account for the status of ongoing ADP security programs in Navy computer systems and networks.

RISK MANAGEMENT CONCEPTS

This session is designed to provide the student a working knowledge of the role of Risk Management as an Accreditation Process and will be able to relate the Risk Assessment, Security Test and Evaluation, and Contingency Planning sub-processes to an Activity-level Risk Management Program.

**ACTIVITY ADP SECURITY PLAN (AADPSP) AND
ACTIVITY ACCREDITATION SCHEDULE (AAS)**

The students are instructed in how to develop a plan for establishing a cohesive ADP Security Program within their activities, utilizing the AADPSP requirements. The plan defines:

1. Scope of the Activity ADP Security Program
2. Commanding Officer's Policy Statement
3. ADP Organization and Responsibilities
4. Objectives of the Activity ADP Security Program
5. Description of the Current ADP Security Environment
6. ADP Security Training
7. Audit/Internal Review
8. ADP Security in Life Cycle Management
9. ADP Security in Configuration Control
10. The Activity Accreditation Schedule (AAS)

The AAS provides a Plan of Action and Milestones (POA&M) for the Accreditation progress of the Activity.

RISK ASSESSMENT (METHOD I)

Risk Assessment (Method I) instructs the students to be able to determine the circumstances in which Method I Risk Assessments must be performed, understand the steps involved in this method and be prepared to organize and manage Risk Assessment studies in their own activities.

CASE STUDY PROBLEM

Within this session the students combine information obtained during an ADP security survey with additional information provided about the scope of operations at a fictitious ADP activity. The members of the class are divided into Risk Assessment Teams and provided information to plan and conduct a Method I Risk Assessment. The workshop presentations are critiqued by the other teams and class solutions are discussed.

ASSETS EVALUATION WORKSHOP

Each team performs asset valuation for one category of assets typical of a Navy ADP activity (such as data, hardware, software, telecommunications, personnel, administrative procedures). The workshop includes asset identification, asset grouping, asset valuation and determining risk assessment impact values in the areas of Modification, Destruction, Disclosure, and Denial of Service.

**THREAT AND VULNERABILITIES EVALUATION
WORKSHOP**

The students conduct threat evaluations for a related set of threats typical of those common to Navy ADP activities. The workshop includes a description of the threat, justification of the vulnerabilities that exist, identification of the existing countermeasures, and estimation of frequency of successful attack for each impact area of Modification, Destruction, Disclosure, and Denial of Service.

ALE DETERMINATION WORKSHOP

The members of the Risk Assessment Teams utilize the results of the asset valuations

and the threat/vulnerability evaluations to calculate an Annual Loss Expectancy (ALE) for the Case Study ADP activity. The ALE determination process is based on the FIPS PUB 65 ALE formula:

$$\text{LOSS} = \frac{\text{IMPACT} \times \text{FREQUENCY OF OCCURRENCE}}{\text{YEARS}}$$

COUNTERMEASURES SELECTION WORKSHOP

The members of the Risk Assessment Teams select and prioritize cost-effective countermeasures to reduce the ALE of the Case Study ADP activity.

RISK ASSESSMENT (METHOD II)

This session instructs the students to perform an abbreviated Risk Assessment using Method II techniques. Method II includes the processes of asset identification and valuation, threat and vulnerability evaluation, ALE computation, and evaluation and selection of additional countermeasures.

This methodology is appropriate for less complex ADP systems and most microcomputer systems.

SECURITY TEST AND EVALUATION (ST&E)

The objective of this session is to provide the students an understanding of Security Test and Evaluation as a component of Risk Management programs and as a part of the accreditation process. The students will be able to plan or conduct an ST&E within their own activities.

CONTINGENCY PLANNING

This session is designed to provide a general knowledge of the Contingency Planning process. This session enables the student to understand how Contingency Plans contribute to Risk Management programs and when they are required for the accreditation of Navy ADP activities and networks.

AUDIT AND COMPLIANCE

The objective for this session is to instruct the students concerning the role of Navy auditors and IG teams in reviewing the security programs of Navy ADP and OIS activities.

ACTIVITY ADP SECURITY TRAINING

This session is designed for the students to understand the activity level training requirements imposed by the DON ADP Security Program and information concerning those alternatives available for meeting the requirements.

CONTRACTING AND REQUESTS FOR PROPOSALS

This lesson enables the students to review the requirements of the contracting needs with COMNAV DAC and understand the role of the ADP security staff in their activity contracting process.

NATIONAL COMPUTER SECURITY CENTER (NCSC)

This session will provide guidelines for the student to review the primary and secondary mission of NCSC, understand the concept of the Trusted Computer System, and describe how the Center can be tasked by DON activities.

COURSE EVALUATION AND QUIZ

The students complete a quiz covering the major issues presented during the course. In addition, students are provided with an opportunity to evaluate the quality and usefulness of course contents.

SUMMARY

What does OPNAVINST 5239.1A require? Your local NAVDAC activity will teach you what you need to know. Come to one of our classes, or arrange for our class at your site, to learn how to :

- Conduct Risk Assessments
- Conduct a Security Test and Evaluation (ST&E)
- Prepare and Test Contingency Plans
- Prepare an Activity ADP Security Plan and Activity Accreditation Schedule (AADPSP/AAS)
- Prepare a Statement of Accreditation
- Obtain Contractor Assistance for ADP Security Compliance
- Obtain Assistance in All of the Above

SOCIAL ASPECTS OF COMPUTER SECURITY

Dorothy E. Denning, Peter G. Neumann, and Donn B. Parker

SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025

Introduction

The problem of computer misuse (intentional and accidental) has been a growing concern as the number of computers and users increases, and as computers become an integral element in areas such as medicine, finance, and defense. This concern has led to advances in computer security technology, and to the Department of Defense Trusted Computer System Evaluation Criteria[1], which gives criteria for evaluating the security of computer systems in terms of the policies to be enforced and the assurance one can obtain in the correct enforcement of those policies. The "Criteria" represents a significant step forward in the computer security area.

The objective of this paper is to examine social aspects of computer security, particularly with respect to some of the technologies being developed. We believe that the problem of computer misuse must be addressed within a broader context that includes the people who regulate and use the system, and the information resources that are external to the system. Security policies and mechanisms must be evaluated in terms of their effect on privacy and productivity, and in terms of the actual and perceived threats they address. If we ignore these social aspects, then there is the danger of developing technologies that are not cost effective, do not address the actual threats, or jeopardize human rights.

In an article on system safety, Leveson [2] observes that "Safety is a system problem," and goes on to show that one cannot make systems safe just by focusing on software and hardware issues. Instead, one must examine the total system and its social aspects, including political, legal, and ethical issues.

The same is true of computer security. We must pay greater attention to the issues of user productivity, privacy, ethics, acceptance of security measures, the nature of the threats, and the role of computer security within the broader context of information security.

In the remainder of this paper, we elaborate on four topics relating to the social aspects of computer security: security policy definition and awareness, user productivity, privacy, and the broad area of information security. For each of these topics, we make specific recommendations aimed at improving overall information security:

Security Policy Definition and Awareness

In many environments, information managers and workers lack the knowledge, motivation, and support to apply basic security controls and practices. This is particularly true in business, where there are few written rules about how the computers may be used. It is not surprising that the systems are misused, because the users and their organizations are not clear what the rules are. Also, many users are not consciously aware of how their carelessness can deleteriously affect other users who are sharing the same resources (including computer networks).

There have been many violations of data privacy and integrity, with a wide variety of motivations - personal gain, greed, curiosity, harassment, etc. Documented cases include external system break-ins; internal fraud and embezzlement; implantation of destructive Trojan horses, software time-bombs inserted for blackmail, spoofing, jamming, and so on. Hiding of knowledge about system security vulnerabilities often (e.g., by system purveyors) creates a head-in-the-sand attitude, ripe for underground dissemination of the vulnerabilities (which are usually known anyway) and abuse. Open discussion of such knowledge also creates problems, as it whets the appetites of would-be perpetrators.

The federal computer crime law and 47 state statutes define as crimes unauthorized acts with, within, or to computers. This makes it imperative for computer systems managers to make clear what is unauthorized, such as personal use of electronic mail and other computer resources. All employees should have explicit requirements to protect information assets in their job descriptions and performance evaluation criteria. Adequate motivation to support security will not be achieved until there are well-defined security policies and until security is considered part of one's job, since security can otherwise be viewed as an obstacle to productivity.

In order to assist organizations develop security policies, policy guidelines can be developed for various types of organizations and various degrees of risk. These guidelines could be developed through industry and professional associations, such as the ACM External Activities Board, the IEEE, and the Data Processing Management Association.

The guidelines would suggest possible policies about what is considered to be acceptable use of an organization's information resources, including personal computers. The policy guidelines should address the broad social issues such as user productivity and privacy rights, discussing tradeoffs as they arise. Based on the guidelines, each organization would formulate its own specific policies in accordance with the sensitivity and value of the information (and other resources) to be protected, and the threats, vulnerabilities, and risks.

When a user is given an account on a system, or other information-related responsibility, the user might be asked to read and sign the organization's policy statement. Making the security policy clear, together with asking all users to make a commitment to the policy, could help eliminate much computer misuse (both internally and externally), while at the same time helping the users appreciate the need for security and the benefits to be gained by it, including making them of greater value to their organizations.

Policies for using personal computers can be developed for elementary and secondary schools. Such policies should contain a clear statement that personal computers are not to be used for unauthorized entry into other computer systems (when in doubt, ask for permission). This could help reduce the malicious hacker problem. Since break-ins are often performed more out of challenge than malicious intent, alternative challenges can be presented to the students in the schools.¹

Overall there is a great absence of pervasive and credible ethical principles; on the other hand, there are many incentives (e.g., weak technology) for violating such a code of ethics, even if it did exist. Nevertheless, such a code should be established, widely taught, and thoroughly practiced within the context of an overall security policy.

Productivity

A main purpose of computers is to aid the productivity of people and organizations. Many users respond negatively towards computer security, because they view it as interfering with their productivity. At least two factors contribute to this attitude: First, many users are not consciously aware of how security helps them with their work, for example, by protecting their files from accidental or malicious destruction, and by allowing selective on-line access to sensitive information. Second, many security mechanisms are overly complicated or tedious to use or install.

In addition, many organizations are reluctant to install security mechanisms that degrade the performance of the system or otherwise interfere with productivity. Indeed, many security mechanisms should not be installed for this very reason, because they are not justified by a cost/risk trade-off. Organizations are also reluctant to switch to more secure systems if the more secure systems are not compatible with the existing systems or provide less functionality. UNIX, for example, has remained popular despite its security weaknesses, because its functional properties con-

¹A recent study on the antisocial behavior of certain members of the computer community [3] concluded that rather different approaches to education are required: "... the cost of these educational environments may be considerably less than the losses being incurred." One particular recommendation was this: "Access to real computing power should be established for interested users, both students and their parents. Empowerment can lead to increased responsibility."

tribute to user productivity. Because of its popularity, several secure versions of UNIX are under development (e.g., see [4]). In many environments, compatibility, performance, and functionality take precedence over security when upgrading to a new system.

If our goal as computer security professionals is to make systems more secure, then we must pay greater attention to the impact of our policies and mechanisms on productivity. In particular, we should strive for policies and mechanisms that, within the scope of threats they address, are transparent to users, simple to install and use, and offer positive benefits to the user community. To illustrate, we will discuss two broad classes of security controls: identification and authentication of users, and discretionary and mandatory access controls.

Identification and Authentication

A variety of different mechanisms has been developed to identify and authenticate users, including passwords, challenge/response protocols, biometrics, keystroke dynamics, access cards, and smart cards. These mechanisms vary considerably both in terms of the security they provide and their impact on productivity. For example, long meaningless *passwords* may offer greater security than short, easy-to-remember ones (if the users do not write them down in obvious locations), but are also more annoying to users. Some security experts have proposed using super-long, but meaningful, passwords, but we do not know whether these are preferred by users over shorter, nonsense passwords, because they require extra key strokes. Moreover, simply lengthening passwords does not protect them from possible exposure during transmission. Cryptographic-based *challenge/response protocols*, such as the PFX system developed by Sytek, can protect against certain threats not addressed by passwords alone (including the exposure threat during transmission), but at the same time lengthen the time required to login. *Biometrics*, such as signature verification, hand geometry, voice prints, and electronic fingerprints can add significant security, but can be expensive and generally require special equipment. Authentication through *keystroke dynamics* is attractive in terms of user productivity, because it is totally passive, low-cost, and transparent, requiring no action on the part of users. In addition, it offers continuous authentication, thereby protecting a user's session while the user is absent from the terminal. On the other hand, because of its passivity, it might raise privacy issues under certain circumstances if the users are not aware of its presence (we will return to this in the next section). *Smart cards* also can provide a high level of security without the need for much user interaction during login, but again require special equipment.

In addition to the various identification and authentication mechanisms, various strategies are applied when a user requests access to a subsystem or remote host. In many environments, the user must supply a separate password for each subsystem or remote host. Because this places an extra burden on the user, these additional passwords are frequently stored on the system, unencrypted, where they are vulnerable to exposure. Mechanisms that provide a high degree of security without requiring any additional information from the user better support the concept that computers are there to aid people.

Access Controls

Discretionary and mandatory (multilevel) access controls can aid productivity by allowing sensitive information that serves the needs of different users to coexist on a single host computer or network. Without adequate host or network access controls, it is necessary both physically and logically to isolate the information, which interferes with a user's ability to access and integrate information. For example, because no commercial system supports a multilevel-secure database system, users who are cleared for information having different access classes (e.g., different sensitivity levels and/or different compartments) cannot access that data from a common database or manipulate it in a single session.

Discretionary access controls are often complicated, making it difficult to grant or revoke access to an individual user, and difficult to understand the implications of doing so. The former is due in part to inadequate user interfaces. For example, on some systems one must remember obscure commands for granting access and even what bit patterns correspond to what access modes! Search-path strategies further complicate matters. The latter is due in part to the inherent limitations of discretionary controls [5,6], and their lack of policy about information flow, including copies of information. The "setuid" facility of UNIX, for example, attempts to provide a mechanism for enforcing the principle of "least privilege," but has dire consequences if not used correctly. Because of the complications associated with discretionary controls, many users, accidentally or intentionally, grant access to all users rather than to those with a need for access.

Network access controls are often inadequate and difficult to analyze. For example, some network facilities have all sorts of special conventions whereby a user can remotely login or copy files from one machine to another without giving a password. However, there is no clear security policy or model underlying the mechanisms, and the result can be total confusion and misapplication of the functionality. Reid [7] describes how intruders broke into a network of UNIX systems by exploiting vulnerabilities in system directories and permission files. These vulnerabilities often arose from shortcuts taken by programmers to improve their own productivity, thus demonstrating the importance of providing secure mechanisms that do not burden the users, and the importance of making users aware of the consequences of break-ins.

Several studies [8,9,10] have shown the value of multilevel, lattice-based policies for controlling direct and indirect (via information flow) access to information of different sensitivities — that is, for enforcing multilevel security. Such policies are relatively easy to understand, avoid the need for users to grant and revoke access, and avoid the inherent limitations of discretionary policies. Moreover, because of their simplicity, it is possible to build systems that enforce multilevel security with a high level of assurance (B3 or A1), and such systems are now becoming commercially available. These systems are based on the concept of a reference monitor or security kernel. Examples include the Honeywell SCOMP and the Gemini GEMSOS [11]. Systems with a lower level of assurance (B1 or B2) could have enormous practical value in environments where the threat is not great, but the simplicity of multilevel security is desirable.

Applications are under development that can exploit the properties of a system enforcing multilevel security. For example, under sponsorship by the U.S. Air Force Rome Air Development Center (RADC), a team at SRI International and Gemini Computers is developing a formal policy model and design for a multilevel-secure database system, which is to be implemented on top of a reference monitor (e.g., GEMSOS) in order to provide A1 assurance [12,13,14]. The development of such applications will enable users to integrate sensitive data of different classifications, thereby improving user productivity.

Although most of the early work on multilevel security was aimed at protecting classified data, some was aimed at protecting sensitive data in the public sector [10], including proprietary and confidential data. Lipner [15] has shown how multilevel policies can be applied to commercial data, and Cohen [16] has argued that such policies help protect against computer viruses. Although we do not claim that multilevel policies and mechanisms can replace discretionary ones, we believe that their potential in the commercial sector has largely been ignored. While some organizations in the public sector have made efforts to classify information, few if any have attempted clearance of their users. Both classification and clearance must be rigorously and comprehensively accomplished in order to obtain the full benefits of multilevel security.

While multilevel security can improve productivity by allowing the integration of sensitive data having different sensitivity markings, if misused, it can inhibit productivity by restricting the flow of information, thereby interfering with the needs for efficient, timely, and effective analysis of information. For example, attempting to eliminate all covert channels in a system improves security, but also impairs communication and the flow of information; similarly, attempting to solve all possible inference and aggregation problems improves security, but makes data integration and analysis more difficult. When security and productivity compete, the appropriate balance can be determined only by examining the particular application environment.

Discretionary access controls are useful as a means of providing a finer granularity of control in order to enforce "need-to-know" constraints within the assigned classifications. However, because they are inherently more complicated and weaker than mandatory ones, they should not be relied upon to control the flow of sensitive information. The limitations of discretionary controls are particularly evident in databases, where access controls may be at the view level (or transaction level) so that authorization can be value-dependent, context-dependent, or history-dependent.

Other types of controls are also needed in order to ensure the consistency or integrity of data, and to enforce other security policies. Our formal model of a multilevel-secure database system, for example, supports database consistency through integrity constraints, transactions, and a mandatory integrity policy [17,18]. Clark and Wilson [19] argue that integrity is more important than multilevel secrecy in most commercial environments, and go on to argue that such a policy should include controls that enforce separation of duty among employees.

Privacy

Computer security is essential for enforcing state and national privacy laws. At the same time, the process of detecting threats, vulnerabilities, and abuses may result in violations of privacy and other human rights, leading to a conflict between the use of computer security to guarantee privacy and its use to invade privacy. These privacy issues became particularly apparent when backup files for a computer operated by the National Security Council were used to reconstruct and expose electronic mail messages regarding the Iran arms deal.

One area where this conflict is especially noticeable is *threat monitoring* — that is, analyzing system activity with the objective of detecting computer break-ins and abuse. We have identified several types of monitoring, listed in order of increasing privacy implications:

1. Continuous authentication, such as through keystroke dynamics.
2. Monitoring unusual activity on the system through system status information (e.g., tracking password failures and looking for sudden rises in system or network activity).
3. Maintaining an audit trail of user activity for the purposes of enforcing user accountability. User events recorded in an audit trail may include login times and locations, commands executed, and file accesses. This type of auditing is required by the Criteria [1] for systems that are rated at the level of C2 and above.
4. Analyzing user events as recorded in an audit trail in terms of abnormal behavior, where "normal" may be defined in terms of a user's past behavior or in terms of acceptable behavior. Under sponsorship from the Space and Naval Warfare Command (SPAWAR), we are developing at SRI a real-time Intrusion-Detection Expert System (IDES) that would detect various types of intrusions by looking for abnormal behavior on the system[20,21].
5. Monitoring the contents of files and messages (e.g., as for the Iran arms case). Any backup system potentially gives a mechanism for implementing this type monitoring, though they are generally not used for this purpose.
6. Complete surveillance of a user's terminal session — i.e., all information transmitted to and from a user's terminal (except possibly passwords). Limited forms of surveillance that provide this type of monitoring have been installed in some systems, and Clyde Digital Systems has developed a surveillance tool called the "Surveillance-Kernel."

Monitoring has many advantages. For example, it has been used to catch outsiders who have broken into computer systems, and it could potentially detect other forms of computer misuse that go undetected by other security controls. Monitoring might be especially attractive in environments where the systems themselves lack adequate security controls commensurate with the sensitivity of the information handled by them. By protecting confidential information

about individuals from unauthorized access, monitoring can help enforce privacy rights and protect information assets.

While recognizing the benefits of monitoring, we have some concern that monitoring could foster a chilling and suspicious attitude in the working environment, especially if it is misused. In particular, the users could feel that they are not considered trustworthy or that their privacy and other rights are violated [22]. We are also concerned that threat monitoring could have an escalating effect as additional monitoring capabilities are developed in order to protect against a wider range of threats, while at the same time the user community becomes increasingly less satisfied with the working environment. Further, monitoring can aggravate the security problem if the data that are accumulated are sensitive but not adequately protected. For example, many audit logs accidentally expose user passwords, such as when a password shows up instead of the user identifier. Finally, the centralization of sensitive audit data that is not otherwise available in an integrated form has social implications.

Because real-time threat monitoring systems are not yet generally available, it is difficult to determine the extent to which these concerns are justified. We can get some insights from a study by Irving, Higgins, and Safayeni [23] on computerized performance monitoring, which showed that "workers perceive increased stress, lower levels of satisfaction, and a decrease in the quality of their relationships with peers and management as a consequence of computerized monitoring." At the same time, however, those authors found that the cause of the dissatisfaction was not so much the monitoring per se as that "managers overemphasize the importance of quantity [over quality] ... in evaluating employee performance." Thus, that study concluded that it is "not the technology itself, but rather how it is used by management that determines an individual's reaction."

We believe that any threat monitoring must be carefully applied to preserve the rights of privacy and freedom from intrusion, and avoid creating an atmosphere that leads to employee and other user dissatisfaction. When a computer system is being shared, users should not expect that they can function privately, in isolation; yet limits must be put on monitoring lest it become oppressive. These issues might be partially resolved by comparisons with analogous situations such as the sanctity of employee's desks and lockers, inter-office mail, television monitoring, use of work-place informants, and telephone eavesdropping practices. If suitably restricted and administered, monitoring of computer activity could be viewed as a benefit by the user community in much the same way that security monitoring of personal luggage at airports is viewed as a benefit by air travellers.

We recommend that a policy be developed regarding threat monitoring that addresses such areas as limits on threat monitoring, use of the results obtained from monitoring, obtaining informed consent of users, and providing due notice of intent to monitor. The development of a monitoring policy should not be limited to security experts, but should involve users, as well as psychologists, sociologists, constitutional lawyers, and human rights groups. We believe that this task should be assigned high priority in order that we do not find ourselves with threat monitoring systems that foster social problems in the work place. Our ultimate

goal must be to create an atmosphere that motivates people to behave responsibly and with confidence that both their rights and information assets are protected.

Protecting Noncomputerized Information

Although our society is still heavily dependent on information that is spoken and printed on paper, we often ignore the security of these other forms of information in favor of the technological challenges associated with automated information. Interviews of approximately 100 computer criminals, while not necessarily representative of all loss experience, indicate a skewing of emphasis across all forms of information [24]. Except for some of the malicious hackers, these people were attempting to solve their intense, unsharable, personal problems with the easiest, safest, and surest methods, constrained by their own skills, knowledge, and resources. Their preferred forms of information were the spoken word first, printed information second, and computerized information third. Computerized information received their focus of attention only when the other forms of information were not accessible to them or amenable to their knowledge and skills[25]. They did not need the computer as a tool to modify, disclose, or manipulate large amounts of information.

Protectors of information must assign similar priorities in applying security, while not overlooking computerized information in anticipation of the few, unusual perpetrators who do not fit the general pattern. Limited security resources would dictate "spoof-proofing" of key employees so that they are not deceived into giving information to outsiders who lack a need-to-know, and protecting printed paper and removable computer media before protecting information stored in computers or data communications [26].

Summary and Recommendations

The pursuit of technology, in the absence of a broad policy that addresses the social aspects of computer security, operates in a vacuum that may lead to violations of human rights, abuse, or other unwanted consequences. Attention must be given to the social aspects, and we make the following specific suggestions:

1. That the social aspects of specific computer security policies and technologies be examined in depth. Areas that should be addressed include identification and authentication, access controls (including those provided by add-on security packages), encryption, and threat monitoring. The technologies should be examined in terms of their actual and perceived effect on productivity and privacy.
2. That generic security policies be developed for different types of organizations and environments, taking into account the social aspects of information protection. The generic policies could serve as guidelines for formulating specific security policies within an organization.

3. That a national policy be developed specifically for threat monitoring that recognizes the rights of the users as well as the potential threats.

Even though the emphasis in this paper is on the social aspects, it is vital that the technological and the social considerations be balanced. They must go hand in hand. Either one without an understanding of the other is likely to create serious problems.

Moreover, security must be tempered with many other requirements that we have not addressed here, such as reliability, safety of use, and real-time responsiveness. To address a broad spectrum of requirements requires a holistic approach. At lower layers of system abstraction we tend to optimize rather locally to ensure that the technology satisfies rather specialized properties such as file privacy and integrity. At the higher layers the optimization may produce completely different results when all of the requirements are considered (technological and human, health and welfare, costs of automating, costs of not automating, etc.)[27].

Acknowledgments

We are grateful to Peter Denning and John Rushby for their constructive comments.

References

- [1] *Department of Defense Trusted Computer System Evaluation Criteria*. Dept. of Defense, National Computer Security Center, Dec. 1985. DOD 5200.28-STD.
- [2] N. Leveson. *Software safety: why, what, and how*. *ACM Computing Surveys*, 1986. to appear.
- [3] J.A.N. Lee, G. Segal, and R. Steier. Positive alternatives: a report on an ACM panel on hacking. *Comm. ACM*, 29(4):297-9, Apr. 1986.
- [4] V. D. Gligor et al. On the design and the implementation of secure XENIX workstations. In *Proc. IEEE 1987 Symp. on Security and Privacy*, pages 102-117, IEEE Computer Society, Apr. 1986.
- [5] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Comm. ACM*, 19(8):461-471, Aug. 1976.
- [6] D. E. Denning. *Cryptography and Data Security*. Addison-Wesley, Reading, Mass., 1982.
- [7] B. Reid. Risks: lessons from the stanford UNIX breakins. *ACM SIGSOFT Software Engineering Notes*, 11(5):29-35, October 1986. A shortened version of this material appears in *CACM* 30(2), February 1987, pp. 103-5.
- [8] C. Weissman. Security controls in the ADEPT-50 time-sharing system. *Proc. Fall Jt. Computer Conf.*, 35:119-133, 1969.
- [9] D. E. Bell and L. J. LaPadula. *Secure Computer Systems: Mathematical Foundations and Model*. Technical Report M74-244, The MITRE Corp., Bedford, Mass., May 1973.

- [10] D. E. Denning. A lattice model of secure information flow. *Comm. ACM*, 19(5):236-243, May 1976.
- [11] R. R. Schell, T. F. Tao, and M. Heckman. Designing the GEMSOS security kernel for security and performance. In *Proc. 8th National Computer Security Conf.*, pages 108-119, 1985.
- [12] D. E. Denning, S. G. Akl, M. Heckman, T. F. Lunt, M. Morgenstern, P. G. Neumann, and R. R. Schell. Views for multilevel database security. *IEEE Trans. on Software Eng.*, SE-13(2):129-140, Feb. 1987.
- [13] D. E. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. Shockley. A multilevel relational data model. In *Proc. 1987 Symp. on Security and Privacy*, IEEE Computer Society, 1987.
- [14] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. Shockley. Element level classification with A1 assurance. 1987. Computer Science Lab, SRI International.
- [15] S. B. Lipner. Non-discretionary controls for commercial applications. In *Proc. 1982 Symp. on Security and Privacy*, pages 2-10, IEEE Computer Society, 1982.
- [16] F. Cohen. Computer viruses, theory and experiments. In *Proc. 7th DOD/NBS Computer Security Conf.*, pages 240-263, Sept. 1984.
- [17] R. R. Schell and D. E. Denning. Integrity in trusted database systems. In *Proc. 9th National Computer Security Conf.*, pages 30-36, 1986.
- [18] D. E. Denning, T. F. Lunt, P. G. Neumann, R. R. Schell, M. Heckman, and W. Shockley. Security policy and interpretation for a class A1 multilevel secure relational database system. Nov. 1986. Computer Science Laboratory, SRI International.
- [19] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *Proc. 1987 Symp. on Security and Privacy*, IEEE Computer Society, Apr. 1987.
- [20] D. E. Denning and P. G. Neumann. *Requirements and Model for IDES - a Real-Time Intrusion Detection System*. Technical Report, Computer Science Laboratory, SRI International, 1985.
- [21] D. E. Denning. An intrusion-detection model. *IEEE Trans. on Software Eng.*, SE-13(2):222-232, Feb. 1987.
- [22] G. T. Marx and S. Sherizen. Monitoring on the job: how to protect privacy as well as property. *Technology Review*, 63-72, Nov./Dec. 1986.
- [23] R. H. Irving, C. A. Higgins, and F. R. Safayeni. Computerized performance monitoring systems: use and abuse. *Comm. ACM*, 29(8):794-801, 1986.
- [24] D. B. Parker. Consequential loss from computer crime. In *Proc. IFIP/Security 86*, 1986.
- [25] D. B. Parker. *Fighting Computer Crime*. Charles Scribner's Sons, New York, 1983.
- [26] D. B. Parker. *Managers Guide to Computer Security*. Reston, Reston, VA, 1981.
- [27] P. G. Neumann. On hierarchical design of computer systems for critical applications. *IEEE Trans. on Software Eng.*, SE-12(9):905-920, Sept. 1986.

SECURITY AND PRIVACY: Issues of Professional Ethics

Dr. Marlene Campbell
Murray State University
Murray, Kentucky 42071

ABSTRACT

The primary purpose of this paper is to provide academicians with both motivation and ideas for bringing ethics formulation into the college Computer Science or Computer Information Systems classroom. It provides some mechanisms for introducing the topic and discussing its importance. It further provides some fundamental facts and documents that are basic to any such discussion.

INTRODUCTION

It was a routine morning as John was on his way to his job in state government. Traffic was not unusually heavy when he was stopped at the light at a busy intersection. For some reason, the occupant of the car next to him caught his eye and his imagination. The car was a Mercedes 190e, and the occupant was a striking young woman about middle thirties. Traffic in her lane moved a bit faster than his, and he was further intrigued by the plate number M I N E .

When he arrived at the office, he did all the routine early morning tasks before flipping on his computer terminal. Once the screen was illuminated and he was logged on to the system, however, he made some very unusual requests for information. By accessing the Department of Motor Vehicles database and entering that plate number, he quickly learned the young woman's name, her address, vital statistics, and driver's license number (which also happened to be her social security number). He then accessed various databases maintained by state and local government and was able to learn the following:

From the State Tax Office he learned her place of employment, her position, and her salary.

From the Tax Assessor's Office he learned the value of her property and the conditions of her deed. (A joint ownership with a different last name signaled that she was divorced.)

From divorce records in the County Clerk's Office, he gained information about her children, and he learned that she had another previous marriage that had terminated in divorce in Reno, Nevada.

He pondered the situation a few moments before making his next move, and he opted for check-

ing school records rather than linking with the State Data Exchange Network to find more details about this first marriage. In checking school records, he learned names, ages, and academic characteristics of her two children.

Before abandoning his quest for information on the young woman, he added her name to a list of legitimate requests for unearned income information from the Internal Revenue Service. Within a few days, he learned that during the previous year she had earned in excess of \$40,000 from investments, rental income, bonuses, and winnings at the track.

All of this was, indeed, enough to convince her that a friend had suggested he contact her when he phoned to make arrangements for a date.

THE FACTS

The story you have just read is not true. It was a scenario set by Pete Early in his article "Prying Eyes" published in the Louisville, Kentucky COURIER JOURNAL MAGAZINE on July 20, 1986. He used this introduction to question the legality of the government's role in creating such database linkage capabilities. I use it to introduce a lively classroom discussion on "Ethics and the Computer Professional."

Computer Security Systems

Did he violate or defeat any computer security system? To answer this question, it is necessary to consider what constitutes a security system.

Figure 1 graphically illustrates the various layers of computer security, the first of which is sound company policies and procedures for access and use. Because this layer is somewhat vague and arbitrary, it is depicted with a broken line. This is a very vulnerable level at which ethics play a very important role.

Other levels include environmental control which is some form of physical isolation, hardware control, software control, and encryption. The first two layers can easily be penetrated if the computer has dial-up capabilities; and the remaining layers, being devised by man, can be defeated by man.

Without knowing the state policies and procedures, we still cannot determine if John violated that level. It is highly likely that he

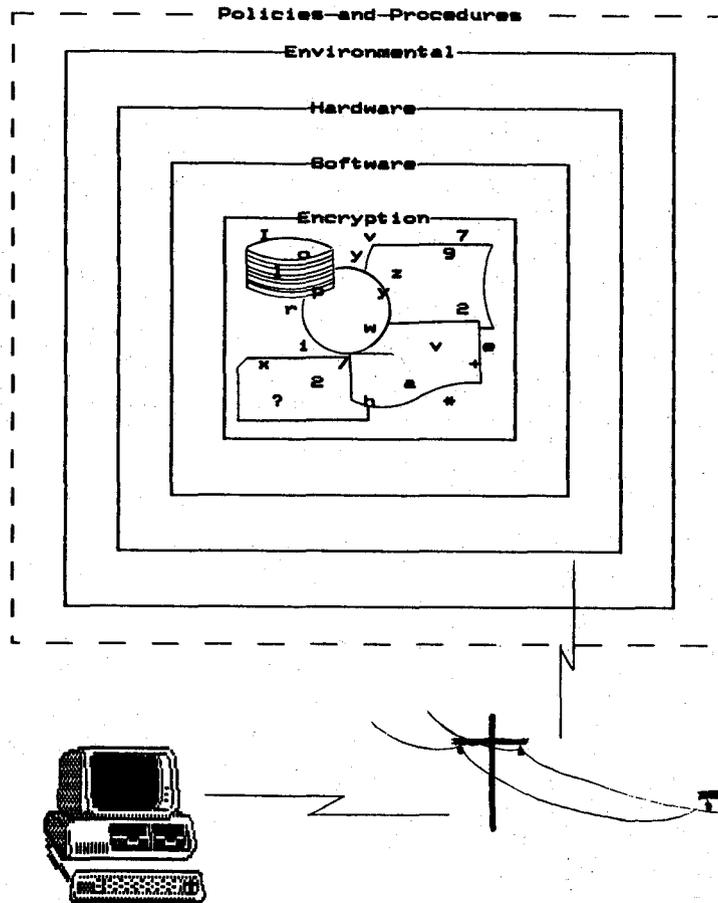


FIGURE 1: LAYERS OF COMPUTER SECURITY

did. We can determine that he did not violate other levels of security. He was an insider who had access to this information but simply chose to use his access capability in a somewhat questionable manner.

Computer Crimes Law

Did he break any laws? Again, before we can answer, we must know where computer crimes laws exist and what they cover.

Since 1978 forty-seven of the fifty states have enacted specific computer crimes legislation. Florida has the oldest law and New York, Texas, and Indiana have the newest. Many of these laws did not come easy.

At the ACM Conference in New York in 1983, there was lively discussion on the pros and cons of such laws. Kenneth Thompson of Bell Labs claimed that the media was causing legislation to start popping up in state legislatures that would impose heavy criminal penalties for unauthorized access to computers that were an unnecessarily harsh response to acts that were more like "computer joy riding." He recommended simple instruction for youngsters that such activities were akin to vandalism and should not be practiced. David Brandon, then President of ACM, charged people who operate unprotected systems with contributory

negligence and cited a need for increased consciousness of their responsibilities [3].

About that same time, Bess Gallanis [5] published an article in which she stated:

...as quickly as security systems are designed, ingenious criminals or precocious kids seem to be that much more challenged to find a weak link in the security chain. Until perfect security is designed, the future lies in pending legislation and court decisions that will define specific crimes and attach appropriate penalties.

John Soma [7] in his book published in that same year wrote:

Although the majority of computer related crimes are basically "the same crimes that have been prosecuted since the apple was plucked," it is difficult to match a specific crime with the traditional criminal statute.

After computer crime hit Congress with the infiltration of computer systems belonging to California Representative Ed Zschau and Arizona Representative John McCain early in 1986 [2], legislative response was strong and swift. Not only did some of the larger states

quickly enact laws, Congress passed two bills to amend Title 18, United States Code. The Computer Fraud and Abuse Act of 1986 which became Public Law No: 99-474 on October 16, 1986 provides additional penalties for fraud and related activities in connection with access devices and computers. The Electronic Communications Privacy Act of 1986 which became Public Law No: 99-508 on October 21, 1986 amended the Federal criminal code to extend the prohibition against the unauthorized interception of communications to include specific types of electronic communications in addition to the interception of wire and oral communication only.

Since he opted not to access the State Data Exchange Network and he did not use a computer to gain information from the IRS, it is reasonable to assume that John did not violate the aforementioned Federal law; but the question still remains as to whether or not he violated a state statute. Since the story originated in Kentucky, consider how he would fare in light of the Kentucky legislation.

The following excerpts from the Kentucky Revised Statutes 434.840 - 434.860 may yield some insight at that level.

KRS 434.840 - 434.860

434.840. Definitions.-- . . .

434.845. Unlawful access to a computer in the first degree. -- (1) A person is guilty of unlawful access to a computer in the first degree when he knowingly and willfully, directly or indirectly, causes to be accessed, or attempts to access any computer software, computer program, data, computer system, computer network, or any part thereof, for the purpose of:

- (a) Devising or executing any scheme or artifice to defraud; or
- (b) Obtaining money, property, or services . . .

(3) Unlawful access to a computer in the first degree is a Class C Felony.

1985 SESSION LAWS OF KANSAS

CRIMES AND PUNISHMENTS

CHAPTER 108 *
Substitute for House Bill No. 2044

AN ACT relating to crimes and punishments; concerning computer crime and unlawful computer access; classifying certain acts as misdemeanors and felonies.

Be it enacted by the Legislature of the State of Kansas:

Section 1. (1) As used in this section, the following words and phrases shall have the meanings respectively ascribed thereto:

- (a) "Access" means to approach, instruct, communicate with, store data in, retrieve data from, or otherwise make use of any resources of a computer, computer system or computer network.
- (b) "Computer" means an electronic device which performs work using programmed instruction and which has one or more of the capabilities of storage, logic, arithmetic or communication and includes all input, output, processing, storage, software or communication facilities which are connected or related to such a device in a system or network.
- (c) "Computer network" means the interconnection of communication lines, including microwave or other means of electronic communication, with a computer through remote terminals, or a complex consisting of two or more interconnected computers.
- (d) "Computer program" means a series of instructions or statements in a form acceptable to a computer which permits the functioning of a computer system in a manner designed to provide appropriate products from such computer system.
- (e) "Computer software" means computer programs, procedures and associated documentation concerned with the operation of a computer system.
- (f) "Computer system" means a set of related computer equipment or devices and computer software which may be connected or unconnected.
- (g) "Financial instrument" means any check, draft, money order, certificate of deposit, letter of credit, bill of exchange, credit card, debit card or marketable security.
- (h) "Property" includes, but is not limited to, financial instruments, information, electronically produced or stored data, supporting documentation and computer software in either machine or human readable form and any other tangible or intangible item of value.
- (i) "Services" includes, but is not limited to, computer time, data processing and storage functions and other uses of a computer, computer system or computer network to perform useful work.
- (j) "Supporting documentation" includes, but is not limited to, all documentation used in the construction, classification, implementation, use or modification of computer software, computer programs or data.

- (2) Computer crime is:
 - (a) Willfully and without authorization gaining or attempting to gain access to and damaging, modifying, altering, destroying, copying, disclosing or taking possession of a computer, computer system, computer network or any other property;
 - (b) using a computer, computer system, computer network or any other property for the purpose of devising or executing a scheme or artifice with the intent to defraud or for the purpose of obtaining money, property, services or any other thing of value by means of false or fraudulent pretense or representation; or
 - (c) willfully exceeding the limits of authorization and damaging, modifying, altering, destroying, copying, disclosing or taking possession of a computer, computer system, computer network or any other property.Computer crime which causes a loss of the value of less than \$150 is a class A misdemeanor.
Computer crime which causes a loss of the value of \$150 or more is a class E felony.
- (3) In any prosecution for computer crime, it is a defense that the property or services were appropriated openly and avowedly under a claim of title made in good faith.
- (4) Unlawful computer access is willfully, fraudulently and without authorization gaining or attempting to gain access to any computer, computer system, computer network or to any computer software, program, documentation, data or property contained in any computer, computer system or computer network.
Unlawful computer access is a class A misdemeanor.
- (5) This section shall be part of and supplemental to the Kansas criminal code.

Sec. 2. This act shall take effect and be in force from and after its publication in the statute book.

Approved April 18, 1985

FIGURE 2: CRIMES AND PUNISHMENT, Chapter 108

Had he been in Pennsylvania, Title 18, Section 3933 UNLAWFUL USE OF COMPUTER states:

(a) OFFENSE DEFINED - A person commits an offense if he:

(1) accesses, alters, damages or destroys any computer, computer system, computer network, computer software, computer program or data base or any part thereof, with the intent to interrupt the normal functioning of an organization or to devise or execute any scheme or artifice to defraud or deceive or control property or services by means of false or fraudulent pretenses, representations or promises; or

(2) intentionally and without authorization alters, damages or destroys any computer, computer system, computer network, computer software, computer program or computer data base or any part thereof.

It is still not clear cut. He did use a computer to access data, but did he have plans to defraud or obtain money, property, or services? We could examine the definition of defraud, a word which occurs in virtually all of the state laws.

DEFRAUD - To deprive of some right, interest, or property by deceit. -- Syn. See CHEAT.

There are many similarities in the various state laws. There are also some very interesting differences in them. North Carolina excludes schemes "to obtain educational testing material, a false educational testing score, or false academic or vocational grade" for consideration as fraud and classification as a felony; and North Dakota specifically mentions "violation of data processing information confidentiality." Georgia says the duty to report violations is coupled with immunity from any civil liability for such reporting.

The Kansas Law, which is not atypical but is somewhat more brief than many, is included in its entirety in Figure 2 as a sample of a complete piece of computer crimes legislation.

Was he guilty of fraud or did he simply satisfy his normal curiosity? We would need to know more of the story, and then we would probably still need to deliberate a very long time.

Privacy Protection

Did he invade her privacy? Surely, this answer is "yes"! But, does she have any protection against this type of invasion of privacy?

Two hundred years ago Thomas Jefferson said:

... laws and institutions must go hand in hand with the human mind... As new discoveries are made, new truths disclosed, and manners and opinions change with the circumstances, institutions must advance also, and keep pace with the times.

In 1890, Supreme Court Justice Louis Brandeis stated that "the right to be let alone is one

of the most comprehensive of rights and the most valued by civilized man."

Now there are, indeed, some federal laws that relate to privacy. The Fair Credit Reporting Act of 1970 which regulates credit bureaus; the Freedom of Information Act of 1970 which permits individuals to have access to data on them contained in federal agency files; the Education Privacy Act which pertains to certain practices of federally funded educational institutions; the Privacy Act of 1974, the introduction of which is contained in Figure 3, which provides certain safeguards for individuals against an invasion of personal privacy by federal agencies; the Right to Financial Privacy Act of 1978 which restricts government access to certain records held by financial institutions; and the Electronic Funds Transfer Acts of 1979 and 1980 which outline responsibilities of companies using EFT are among them. These laws, however, govern only the actions of federal agencies or agencies who receive funds from the federal government.

Many of the states also have laws relating to privacy, and these laws were quite often enacted well in advance of computer crimes laws. Again, however, most do not specifically cite invasion of privacy as it relates to computers and databases; and interpretation of guilt under such laws would be uncertain.

Few information privacy violation cases have been litigated. Since we do not know what is accumulated, stored, and transferred pertaining to us and it is not likely that we will ever know, we are unlikely to pursue such a matter. If we go to court to protect our privacy everyone will know everything. Is the gain worth the risk? Probably not.

If accused, then, could John be convicted of invasion of privacy? That, too, is doubtful.

THE FUNDAMENTALS

Professional Ethics

Would his actions be considered ethical among computer professionals? Before deciding, we should consider statements pertaining to ethics in the Computer Science and Computer Information Systems literature.

Definitions

ETHICS - A treatise on morals; the science of moral values and duties; the study of ideal human character, actions, and ends; moral principles, quality, or practice. (Webster's New Collegiate)

ETHICS - A system of moral principles; the rules of conduct recognized in respect to a particular class of human actions or to a particular group, culture, etc.; moral principles; that branch of philosophy dealing with values pertaining to human conduct with respect to the rightness and wrongness of certain actions and to the goodness and badness of the motives and ends of such actions. (Random House Unabridged)

PRIVACY ACT OF 1974

INTRODUCTION

The purpose of this act is to provide certain safeguards for individuals against an invasion of personal privacy by requiring Federal agencies, except as otherwise provided by law, to:

1. Permit an individual to determine what records pertaining to him are collected, maintained, used or disseminated by such agencies.
2. Permit an individual to prevent records pertaining to him obtained by such agencies for a particular purpose from being used or made available for another purpose without his consent.
3. Permit an individual to gain access to information pertaining to him in federal agency records, to have a copy of all or any portion thereof, and to correct, or amend such records.
4. Collect, maintain, use or disseminate any record of identifiable personal information in a manner that assures that such action is for a necessary and lawful purpose, that the information is current and accurate for its intended use, and that adequate safeguards are provided to prevent misuse of such information.
5. Permit exemptions from the requirements with respect to records provided for in this act only in those cases where there is an important public need for such exemption as has been determined by specific statutory authority.
6. Be subject to civil suit for any damages which occur as a result of willful or intentional action which violates any individual's rights under this act.

The PRIVACY ACT OF 1974 applies to all federal agencies except the CIA and law enforcement agencies.

FIGURE 3: PRIVACY ACT OF 1974

We live in an age and in a society where morality and ethics seem to be eroding more each day. We are amazed when we read the ever growing accounts of questionable behavior on the part of prominent and not so prominent members of our society. We shake our heads and role our eyes when we read of unethical philanderings, but we do little or nothing definitive to bring about significant change in such behaviors. We read and relate the accounts which glorify such behavior, and we buy tickets and attend lectures to hear former convicted felons present the rationale for their unethical and/or criminal behavior.

In May 1984, COMPUTERS and PEOPLE carried an article titled "Lack of Ethics as a Cause for Computer Crime" which was excerpted from Chapter 6 of HOW TO PREVENT COMPUTER CRIME by August Bequai. It exhibited the Table of Contents of the book and then began:

Lack of Ethics

A computer programmer attempts to sell valuable software belonging to his employer to one of its competitors. When discovered, the employer is reluctant to prosecute; it is rumored that the programmer threatened to "blow the whistle" on corrupt company practices. An executive embezzles more than \$400,000 of his company's assets through the use of its computer. When the auditors uncover his fraud, his employer simply asks for his resignation; it is said that dishonest conduct was a "way of life" at the company. A computer operator uses a hospital's computer to steal more than \$20,000; the victim is reluctant to prosecute. It is alleged that an investigation would have led to exposure of thefts of drugs involving hospital personnel.

The above examples serve to illustrate two key points: first, that crime by dishonest employees has reached epidemic proportions ... ; and second, that some victims are reluctant to prosecute because they have their own "skeletons" to hide. ...

Other subheadings included:

- Law of the Jungle
- Glorifying the Computer Criminal
- We Ill-Treat "Whistleblowers"
- Role of Ethics
- Need for Ethical Management
- Implementing a Code
- A model Code
- Testing your Code

He cited a study by the Ethics Resource Center of Washington, D.C., that "confirmed that, although top management occasionally pays lip service to the need for ethics in the workplace, little is done to carry this out." [1]

Robbin Juris [6], Associate Editor of COMPUTER DECISIONS, claimed in his article "Keeping Out the Insiders" that "security breaches by outsiders may be obscuring a much greater risk to corporate computer systems: the threat from within." Only clearly stated company policies and procedures and ethical conduct of legitimate users will eliminate, or at least reduce, such security breaches from inside.

Both the Association for Computing Machinery (ACM) and the Data Processing Management Association (DPMA) have made positive and definitive statements pertaining to professional ethics. Excerpts from BYLAW 19, ACM CODE OF PROFESSIONAL CONDUCT are included in Figure 4. The entire DPMA Code of Ethics is presented in figure 5.

Note that the ACM code begins:

PREAMBLE

Recognition of Professional Status by the public depends not only on skill and dedication but also on adherence to a recognized Code of Professional Conduct.

DPMA chooses to begin its CODE OF ETHICS with a statement of obligation to management, to fellow members, to society, to employer, and to country. The STANDARDS OF CONDUCT, then, specify the responsibilities of each of these obligations.

In light of these presentations, was he guilty of a breach of ethics? Did he violate any subsections of either CANON 1 or CANON 5 of the ACM code? Is there an infraction against the DPMA code? If his activities were known, would he be disciplined for his actions by one of these professional associations? As computer professionals, how should we react to such behavior in our ranks?

Instructional Activities

Virtually every college textbook for use in introductory information systems courses has one or more chapters devoted to social issues and implications. Most have at least a brief discussion relating to ethics. Texts for introductory computer science courses contain chapters on file processing and occasionally make reference to data security and integrity.

BYLAW 19, ACM CODE OF PROFESSIONAL CONDUCT

PREAMBLE

Recognition of professional status by the public depends not only on skill and dedication but also on adherence to a recognized code of Professional Conduct. The following Code sets forth the general principles (Canons), professional ideals (Ethical Considerations), and mandatory rules (Disciplinary Rules) applicable to each ACM Member.

The verbs "shall" (imperative) and "should" (encouragement) are used purposefully in the Code. The Canons and Ethical Considerations are not, however, binding rules. Each Disciplinary Rule is binding on each Member of ACM. Failure to observe the disciplinary rules subjects the Member to admonition, suspension or expulsion from the Association as provided by the Procedures for the Enforcement of the ACM Code of Professional Conduct, which are specified in the ACM Policy and Procedures Guidelines. The term "member(s)" is used in the Code. The Disciplinary Rules apply, however, only to the classes of membership specified in Article 3, Section 5, of the Constitution of the ACM. [i.e. members with voting rights]

CANON 1

An ACM member shall act at all times with integrity.

Ethical Consideration

EC1.1 An ACM member shall properly qualify himself when expressing an opinion outside his areas of competence. A member is encouraged to express his opinions on subjects within his area of competence.

EC1.2 An ACM member shall preface any partisan statements about information processing by indicating clearly on whose behalf they are made.

EC1.3 An ACM member shall act faithfully on behalf of his employers or clients.

Disciplinary Rules

DRI.1.1 An ACM member shall not intentionally misrepresent his qualifications or credentials to present or prospective employers or clients.

DRI.1.2 An ACM member shall not make deliberately false or deceptive statements as to the present or expected state of affairs in any aspect of the capability, delivery, or use of information processing systems.

DRI.2.1 An ACM member shall not intentionally conceal or misrepresent on whose behalf any partisan statements are made.

DRI.3.1 An ACM member acting or employed as a consultant shall, prior to accepting information from a prospective client, inform the client of all factors of which the member is aware which may affect the proper performance of the task.

DRI.3.2 An ACM member shall disclose any interest of which he is aware which does or may conflict with his duty to a present or prospective employer or client.

DRI.3.3 An ACM member shall not use any confidential information from any employer or client, past or present, without prior permission.

CANON 2

An ACM member should strive to increase his competence and the competence and prestige of the profession.

CANON 3

An ACM member shall accept responsibility for his work.

CANON 4

An ACM member shall act with professional responsibility.

CANON 5

An ACM member should use his special knowledge and skills for the advancement of human welfare.

Ethical Considerations

ECS.1 An ACM member should consider the health, privacy, and general welfare of the public in performance of his work.

ECS.2 An ACM member, whenever dealing with data concerning individuals, shall always consider the principle of the individual's privacy and seek the following:

- * To minimize the data collected.
- * To limit authorized access to the data.
- * To provide proper security for the data.
- * To determine the required retention period of the data.
- * To insure proper disposal of the data.

Disciplinary Rules

DR5.2.1 An ACM member shall express his professional opinion to his employers or clients regarding any adverse consequences to the public which might result from the work proposed him.

FIGURE 4: ACM CODE OF PROFESSIONAL CONDUCT (excerpts)

Database and file processing courses, system design and analysis, and others have very good entry points for in-depth discussions relating to ethics and the computer professional.

Introducing the topic at a reasonable point in the overall course plan yields an acceptance of the relevance of such a discussion in the classroom. It is then up to the instructor to

capitalize on that moment. A scenario such as presented in this paper provides an excellent springboard. We can relate to this situation. We can put ourselves in the position of either the young man or the young woman. We can reflect on what we would do in a similar situation and what our reaction would be to the possibility that someone could gain so much information about us so readily.

CODE OF ETHICS

I acknowledge:

That I have an obligation to management, therefore, I shall promote the understanding of information processing methods and procedures to management using every resource at my command.

That I have an obligation to my fellow members, therefore, I shall uphold the high ideals of DPMA as outlined in its international Bylaws. Further, I shall cooperate with my fellow members and shall treat them with honesty and respect at all times.

That I have an obligation to society and will participate to the best of my ability in the dissemination of knowledge pertaining to the general development and understanding of information processing. Further, I shall not use knowledge of a confidential nature to further my personal interest, nor shall I violate the privacy and confidentiality of information entrusted to me or to which I may gain access.

That I have an obligation to my employer whose trust I hold, therefore, I shall endeavor to discharge this obligation to the best of my ability, to guard my employer's interests, and to advise him or her wisely and honestly.

That I have an obligation to my country, therefore, in my personal, business and social contacts, I shall uphold my nation and shall honor the chosen way of life of my fellow citizens.

I accept these obligations as a personal responsibility and as a member of this association, I shall actively discharge these obligations and I dedicate myself to that end.

STANDARDS OF CONDUCT

These standards expand on the Code of Ethics by providing specific statements of behavior in support of each element of the Code. They are not objectives to be strived for, they are rules that no professional will violate. It is first of all expected that information processing professionals will abide by the appropriate laws of their country and community. The following standards address tenets that apply to the profession.

In Recognition of My Obligation to Management I Shall:

- * Keep my personal knowledge up-to-date and insure that proper expertise is available when needed.
- * Share my knowledge with others and present factual and objective information to management to the best of my ability.
- * Accept full responsibility for work that I perform.
- * Not misuse the authority entrusted to me.
- * Not misinterpret or withhold information concerning the capabilities of equipment, software, or systems.
- * Not take advantage of the lack of knowledge or inexperience on the part of others.

In Recognition of My Obligation to My Fellow Members and the Profession I Shall:

- * Be honest in all my professional relationships.
- * Take appropriate action in regard to any illegal or unethical practices that come to my attention. However, I will bring charges against any person only when I have reasonable basis for believing in the truth of the allegation and without regard to personal interest.
- * Endeavor to share my special knowledge.
- * Cooperate with others in achieving understanding and in identifying problems.
- * Not use or take credit for the work of others without specific acknowledgement and authorization.
- * Not take advantage of the lack of knowledge or inexperience on the part of others for personal gain.

In Recognition of My Obligation to Society I Shall:

- * Protect the privacy and confidentiality of all information entrusted to me.
- * Use my skill and knowledge to inform the public in all areas of my expertise.
- * To the best of my ability, insure that the products of my work are used in a socially responsible way.
- * Support, respect and abide by the appropriate local, state, provincial, and federal laws.
- * Never misrepresent or withhold information that is germane to a problem or situation of public concern nor will I allow any such known information to remain unchallenged.
- * Not use knowledge of a confidential or personal nature in any unauthorized manner or to achieve personal gain.

In Recognition of My Obligation to My Employer I Shall:

- * Make every effort to ensure that I have the most current knowledge and that proper expertise is available when needed.
- * Avoid conflict of interest and insure that any employer is aware of any potential conflicts.
- * Present a fair, honest, and objective viewpoint.
- * Protect the proper interests of my employer at all times.
- * Protect the privacy and confidentiality of all information entrusted to me.
- * Not misrepresent or withhold information that is germane to the situation.
- * Not attempt to use the resources of my employer for personal gain or for any purpose without proper approval.
- * Not exploit the weakness of a computer system for personal gain or personal satisfaction.

FIGURE 5: DPMA - CODE OF ETHICS and STANDARDS OF CONDUCT

Library assignments followed by group discussion of the most interesting articles is especially beneficial. A note of caution, however. The instructor must be sure where he/she stands on such issues and be ready to field very pointed questions about personal values. The day after such a discussion in one of my classes a student asked me about my feelings on copying software. This could be very touchy, but they deserve honest answers.

Congressman Zschau [2] was quoted as saying that the infiltration of his computer system was "tantamount to someone breaking into my office, taking my files and burning them," but there was no physical evidence of such a break-in and fire. "Because people don't see the files overturned or a pile of ashes outside the door, it doesn't seem so bad. But it is equally as devastating."

Interacting with colleagues to discuss such topics is a great mechanism for building our own values and accumulating 'for instances' to relate to students. We had a faculty dutch treat lunch recently to discuss ethics. I attended a breakfast at a recent professional meeting devoted to this topic. Listening while others talk about touchy issues is quite enlightening.

If there were a file cabinet located in a hallway, should a passerby look in? If there were names on the drawers, would he/she be more or less likely to want to look in? If the cabinet were locked, would he/she feel challenged to look in?

Is there any difference in walking around in Neiman-Marcus at 2 a.m. and walking around in someone's database at 2 a.m.? If caught, would the perpetrators be treated differently? Should they be? It is interesting that the Virginia computer crimes legislation includes a statement to attest that a "tangible document need not be evident when a computer is the instrument of forgery." Do our written laws really need to be this specific?

CONCLUSION

Computers are powerful tools at the disposal of men. Men who understand and use these machines have power to accomplish great and worthy goals or to wreak havoc and destruction. Although security mechanisms and laws are provided to temper the activities of men when they sit down at the machines, the only truly binding controls are the professional ethics of the man.

REFERENCES CITED

1. Bequai, August. "Lack of Ethics as a Cause of Computer Crime." COMPUTERS and PEOPLE (May-June, 1984) 7-14, 24.
2. "Computer Crime Hits Congress." MICRO Matters 1 (May, 1987) 2.
3. "Computer Hacking and Security Costs." Science News 124 (November 5, 1983) 294.
4. Early, Pete. "Prying Eyes." The Courier Journal Magazine (July 20, 1986).
5. Gallanis, Bess. Computer Bandits, The New Brain Pickers." Advertising Age 54 (November 14, 1983) M-32.
6. Juris, Robbin. "Keeping Out the Insiders." COMPUTER DECISIONS (November 4, 1986) 48-49.
7. Soma, John T. Computer Technology and the Law. New York: Shepard's/McGraw Hill, 1983.

DATA INTEGRITY VS. DATA SECURITY: A WORKABLE COMPROMISE

Ronda R. Henning and Swen A. Walker

National Computer Security Center
Office of Research and Development
9800 Savage Road
Fort George G. Meade, Maryland 20755-6000
301-859-4488

INTRODUCTION

There are many diverse opinions on the relative importance of data integrity when compared with the relative importance of data secrecy. In the Department of Defense Trusted Computer System Evaluation Criteria, integrity is defined as the correct operation of the hardware and firmware upon which the operating system's Trusted Computing Base (TCB) resides, and the assurance that the TCB software has not been subject to unauthorized modification. The correct operation of the TCB only ensures that the file system is intact and that the TCB has not been unintentionally or maliciously modified. The Criteria makes no claim as to the validity or consistency of the information that may be contained in the files protected by the TCB.

The problems of data integrity have always existed in trusted operating systems. Data management applications of these systems make the problem more acute. Consistent, accurate, reliable information is a critical element for data management applications. This paper provides an overview of data integrity concerns, and why they are not sufficiently addressed by conventional secrecy policies. The issue of unauthorized modification of data which might compromise data validity is addressed, as is the practicality of the implementation of the current state of the art in integrity policies. The discussion concludes with an attempt to provide guidance on the application of integrity policies to trusted data management.

THE INTEGRITY PROBLEM

Before the Industrial Revolution, most business enterprises were established by and run as single person firms. Data integrity was not a consideration because every piece of information required to run the business came through the proprietor. Industrialization brought larger conglomerates into being. It was not possible to run a nationwide railroad, for example, with one bookkeeper and one accountant. As more people became involved with corporate information, data integrity became a greater problem. Two-man control of

sensitive information was introduced as one control methodology. Single-point access to company filerooms was another way to control access to corporate information. Information, however, was becoming more and more accessible to larger numbers of people as part of their daily duties. However, the amount of information accessible and changeable on any given day was still relatively limited.

The Computer Age increased the data integrity problem significantly. More data was accessible to more people than had ever been possible with manual processing methods. It was also easier than ever before to make wholesale changes to information, such as cleaning out bank accounts, embezzling funds, and just simple system failures wiping out files. Backup copies could replace lost files, but changes were harder to trace to a single individual and restore. Eventually, electronic audit files were used to establish a chain of accountability for modifications. This provided a way to know who last accessed a file or a particular account. It did not offer a remedy to the data entry clerk who mistyped 1.000 instead of 1,000 and forgot to proofread the screen before pressing the transmit key.

Database management systems made it even easier to modify large quantities of data quickly and efficiently with simple query languages. These systems also brought new mechanisms, such as data dictionaries and semantic constraints, into common use as control mechanisms for data integrity [8]. For example, it was now possible to require only numeric data for social security numbers, and social security numbers had to have nine digits. When such mechanisms clashed with performance requirements, however, they were often ignored or circumvented, leaving information more vulnerable to integrity compromise.

DEFINITIONS OF INTEGRITY

This brief history of the integrity problem makes it easy to see that there can be many definitions of data integrity, all of which are valid. Perhaps an integrated definition of integrity can be found in [16]. This definition covers six areas:

- a. How correct we think the information is,
- b. How confident we are that the information is from its original source,
- c. How correct the functioning of the process is,
- d. How closely the process function corresponds to its designed intent,
- e. How confident we are that the information in an object is unaltered, or was correctly modified, and
- f. How correct the information in an object is.

How correct we think the information is can be considered the conventional data management definition of integrity. In the traditional data management environment, integrity is defined as the consistency or validity of data entered into a database or a file. This definition includes semantic integrity constraints which can be specified as part of a data dictionary or application program, such as all salaries must be greater than zero; concurrency controls which ensure that the serializability of transactions is maintained to prevent interference between two or more executing transactions; and recovery mechanisms to ensure the proper restoration of data in the event of a system failure.

How correct the functioning of a process is, and how closely it equates to its designed intent can be defined as operational consistency and correctness. Operational consistency equates to confidence in achieving the same results if the same code is executed repeatedly with the same input. It is the ability to rely on system operations and services, such as daemons, to run properly with predictable results. Correctness is the assurance or guarantee that the system will perform as its designers intended it to and that it will operate properly. This type of integrity is often referred to as system integrity.

How confident we are that the information is from its original source, has been subjected to only authorized modifications, and is correctly represented in a storage object can be considered the computer security definition of integrity. This definition includes the mapping of information into digital data for storage in an object, user authentication, authorization of the user to perform modifications, and confidence that the user entered error-free information into the system which was not maliciously or unintentionally altered by either another user or the operating system.

For example, a user enters data at a terminal. The characters are translated into ascii code, sent to an i/o buffer, and eventually stored in a file. The user is confident that the characters he entered were accurately represented and not altered by a short circuit into a different bit pattern. He may wish to let another user edit this data at a later date, and sets copy privileges on the data for another user. When a third user tries to copy the data, he

is not authorized to modify it and is denied access. If the second user tries to overwrite the data he is not permitted to do so, because it would be an unauthorized modification. The owner of the data also believes that the operating system will not lose his file in the event of a system crash.

WHY WORRY ABOUT INTEGRITY?

Is a trusted computer system that enforces a global security policy as described in the Criteria, sufficient to cope with data integrity concerns? In the Shirley and Schell paper on validation by assignment [20], the argument is made that a security policy is based on external laws, rules, regulations, and other mandates that establish what access to data is permitted. Access to data is defined as what information may be disclosed to any given user, not as what information may be modified by any given user.

However, a security policy that addresses only the disclosure of information is not a complete policy. In Denning and Schell [11], two principal components of the information security policy are proposed: a secrecy class to control information disclosure, and an integrity class to control the modification of information. A trusted system is trusted to protect "sensitive" information from unauthorized disclosure, alteration, or destruction. Therefore, a security policy that addresses only secrecy is not sufficient unless it also addresses data modification issues, or data integrity.

There are precedents and true, paper-based procedures upon which it is possible to model an information disclosure policy. Landwehr [15] has argued that a similar model for an integrity policy does not exist. He states that the government possesses large amounts of sensitive information that would compromise national security if it was revealed to certain organizations outside of the government. Therefore, it has had to institutionalize a protection policy of hierarchical classifications and compartments for this information. No damage assessment of the consequences of unauthorized modification of such information has resulted in a similar set of hierarchical integrity labels. Therefore, a justification for a global integrity policy to protect against unauthorized modification of sensitive data does not currently exist in government regulations. On an application-specific basis, information that must be protected from unauthorized modification can be identified and should be protected using means appropriate to the sensitivity of the information. There is no global integrity policy for the Federal Government.

Beyond these arguments, security policies specified by the Criteria are required to have mechanisms available to ensure the "correct" operation of the software and firmware comprising the TCB. The Criteria further requires that assurances are in place to ensure that the software TCB is subject to sound configuration management practices and distribution techniques. There is no requirement in the Criteria to enforce data consistency constraints or other such

common integrity measures. Therefore, a system meeting the intent of the Criteria does not guarantee the validity of the information represented within its objects.

DOES SECRECY OFFER SOLUTIONS?

Can integrity concerns be addressed through the use of security mechanisms designed to address secrecy? Secrecy concerns, as stated above, address information dissemination, not information modification. Integrity is often considered the dual of secrecy. Using a secrecy lattice for integrity enforcement, therefore, will protect high-level objects from low-level subjects, but equates relative integrity with relative secrecy. That is, the most widely disseminated data (unclassified data) would be considered the least vulnerable to unauthorized modification. The least widely disseminated data (top secret) would be the most vulnerable to unauthorized modification because it would be the most enticing to someone trying to penetrate the system. This is a restatement of the secrecy policy for the system.

A program integrity policy [20] will protect against insertion of malicious code into an application and will ensure the enforcement of integrity constraints upon a user's application. Such a policy assumes that the development staff is trusted. The policy will not enforce an access control policy by itself, and, therefore, cannot be used to enforce authorized modification rules. A program integrity policy does not necessarily have access control lists associated with it to determine which users are authorized to access data and which are not. It can ensure that programs behave "correctly", but cannot ensure against data value corruption by malicious users.

A discretionary integrity policy based on access control lists will limit the modification rights granted to a user. However, it makes no promises about the enforcement of integrity constraints or the "correctness" of code. A discretionary integrity policy is also not automatically or uniformly invoked for each data access by every user. It is not, therefore, a mechanism that is enforceable with a high degree of assurance.

THE PROBLEM OF UNAUTHORIZED MODIFICATION

Various extensions and alternatives to the current generation of secrecy policies have been developed in an attempt to address integrity considerations. Biba [1] discussed three types of hierarchical integrity policies: (1) strict integrity, (2) ring policy integrity, and (3) low water mark integrity.

The strict integrity policy considered integrity the dual of secrecy. Whereas the standard Bell-LaPadula security policy permits the reading down and writing up of information in secrecy classes, Biba contends that these actions compromise the integrity properties of the data. A higher-secrecy-level user compromises his higher-secrecy-level data by the act of reading data at a lower-secrecy-level. A lower-secrecy-level

user performing a blind write to a higher-secrecy-level object may unintentionally or maliciously provide misinformation to a higher-secrecy-level process that wished to use the same data file. In the strict integrity policy, integrity levels are used to counter this threat. A user may read an object if his integrity level is dominated by the object's integrity level. A user may write to an object if his integrity level dominates the object's integrity level.

The ring policy variant on Biba's strict integrity policy states that no restrictions are placed on the reading of data, but the constraints on writing to an object are the constraints specified in the strict integrity policy. A subject may write high integrity data to a file even though he has read low integrity data in the same process.

The object's integrity level is never changed in the low water mark integrity policy, but the integrity level of the subject is degraded when he reads data at a lower integrity level. The subject may eventually have his integrity level decreased to the lowest integrity level on the system, the low water mark. To restore his process to a higher integrity level would require reinitialization of his privileges; in all probability, this would require a trusted process.

Denning and Schell [11] proposed a more flexible variation on the strict integrity policy. This integrity policy proposes that trusted subjects can read objects or write to them as long as they fall within the permitted range of integrity levels. Untrusted subjects are limited to reading or writing objects as stated in Biba's strict integrity policy. An additional constraint is added by the execute property, which states that a subject can execute an object only if the maximum integrity level of the subject is less than or equal to the integrity class of the object, and the maximum secrecy level of the subject is greater than or equal to the secrecy class of the object.

The variation of Biba's strict integrity policy proposed by Shirley and Schell [20] allows the reading of lower integrity level data by higher integrity level processes. Execute access is established as a separate access right, and a process may only execute processes with an equal or greater integrity level. Write access rights are applied as specified in Biba's policy model. This affords a greater degree of flexibility than the strict integrity model because read and update operations could be performed by high-integrity level processes across all lower-integrity levels.

Changeable subject integrity levels and program integrity levels are used in Boyun's [4] variation of strict integrity. As the user reads lower-integrity data, his integrity level is downgraded. This variation also introduces the notion of programs having an integrity level based on their potential to corrupt higher-integrity data. Since they offer the potential for data corruption, programs must have a higher integrity level than the data objects they

will be acting upon. Unfortunately, as the user reads lower and lower integrity level data, his own integrity level is permanently degraded, preventing him from ever reading high-integrity data again. The only way to restore his integrity level is through a trusted upgrade process, executable only by a trusted subject, as in the low water mark policy.

Another type of integrity policy commonly used in commercial applications is enforced strictly through rule-based constraints. These constraints are either written for each application separately or generalized into a global integrity policy for all applications of a particular type. For example, a common set of type-checking utilities for a database management system. This type of policy requires a trusted process to place the user-written constraints into the TCB. Such an expansion of the TCB would, in all probability, make it larger than current analysis techniques could handle, which would make it very difficult to guarantee the level of assurance required at higher levels of the Criteria.

An alternative to Biba's strict integrity policy variations was proposed by Boebert [2] and is being implemented in the LOCK project. This policy uses the concept of type-domain enforcement to implement constraints. Objects are associated with types, and subjects are associated with domains. An access matrix of domains and types determines the rights available to a subject requesting access to an object. The access matrix consists of a combination of static access constraints coupled with application-specific access constraints specified by a trusted user. If a domain does not have access to a given type as specified by the access matrix, it violates the integrity policy and access is not permitted. The type-domain enforcement integrity policy is orthogonal to the secrecy policy and is logically "anded" with the access rights granted by the secrecy policy to determine the subject's effective access rights. That is, the intersection of the two matrices form the access privileges permitted under the system security policy.

ARE THESE POLICIES USEFUL?

Are any of the various integrity policy alternatives practical in a operational environment? Or is the current generation of integrity policies overly protective?

It is important to note that integrity policies are not usually implemented by themselves. They are usually implemented in conjunction with a secrecy policy to provide a system security policy. To determine the utility of an integrity policy, its position must be taken into consideration with respect to the overall system security policy.

Using this criteria, the strict integrity interpretation proposed by Biba does not appear flexible enough to be useful in practical applications, such as database management systems [2]. Applications must have read and write access to various system tables and internal data structures in order to perform their functions. In the context of Biba's strict integrity policy, this can only be accomplished if the integrity level

of all relevant data is system low. However, a system low integrity level affords this data minimal protection under the integrity policy.

The ring integrity policy uses fixed integrity labels on both subjects and objects. It allows the subject, however, to read data at any integrity level and write to objects of lesser or equal integrity levels. No execute access is defined in this policy. Therefore, the subject integrity level is of little value since programs of dubious integrity may be executed by a subject with a high integrity level. Such a practice would allow the destruction of any higher-integrity data which the subject may access. This policy has never been implemented in practice.

The low water mark policy allows a subject to paralyze itself. While necessary objects can be created at higher- integrity levels and the subject can access them as long as it maintains an equal-integrity level, reading a lower-integrity object will degrade the integrity level of the subject, making the higher-integrity objects inaccessible to the now lower- integrity subject.

Denning and Schell's integrity range, Shirley and Schell's program integrity policy, and Boyun's changeable integrity policies all are attempts to incorporate more flexibility into the strict integrity policy. Whether the increased flexibility they provide will also increase the size of the TCB beyond analytical limits, degrade performance, or create new integrity concerns and security covert channels has yet to be investigated.

A rule-based integrity policy is not, in and of itself, applicable to the general case. Rules change from one application to another, and one user interface to another. Additionally, such a rule-based integrity policy may provide a substantial inference channel if simultaneously enforced at multiple secrecy levels with a single instantiation of the data. A user at a given level may be able to carefully construct queries that would allow him to determine the information he was not permitted to access.

The type-domain enforcement mechanism proposed by Boebert may prove useful when the number of enumerated types enforced is relatively small. However, user-specified types may be more numerous and could possibly cause serious performance penalties, unless matrix compression techniques were used on the access matrix. More investigation must be conducted to determine the properties associated with type-domain enforcement mechanisms.

OTHER INTEGRITY CONSIDERATIONS

Polyinstantiation [11] has been proposed as one solution to the integrity-secrecy dilemma. If a higher secrecy or integrity user attempts to perform a modification operation on lower level data, another copy of the data is automatically created that is identical to the original in all but the level designation. This higher-level instantiation of the data then reflects the

higher-level user's modifications. While it will ensure that the security policy is not violated, polyinstantiation will not solve the problems of data consistency. Data that is replicated at each level will guarantee consistency within one level, but will not ensure consistency across levels. The entire database would not necessarily be consistent. In this case, one can never be certain which instantiation of the original data is the most accurate or recent representation.

As a final consideration, one must examine the user interface issues entailed by the various policies. If the user is only permitted to read or write at a single level, mechanical cut-and-paste techniques will be required to obtain all of his data in a single report. Users in general may be able to adjust to almost anything, provided they do not perceive duplication of effort. When update operations become excessively tedious, as may be the case with the strict integrity policy, users become increasingly rebellious and either cease to use the system or develop their own integrity policy: the high water mark. No one piece of the integrity puzzle fits all the empty places in a security policy.

CAN HIGH DATA INTEGRITY EXIST WITH HIGH TRUST?

Data integrity cannot be ignored in a trusted computer system. Assurance that the TCB is functioning correctly at the operating system level is not enough for trusted applications that require data to be valid and consistent. No one data integrity policy clearly satisfies the many varieties of integrity considerations. Environmental factors must also be taken into consideration. Just as the various evaluation classes of the Criteria do not unilaterally apply to all cases, neither can integrity policies be blindly applied without consideration of the sources, frequency of modification, sensitivity, and perishability of data.

A layered approach to data integrity may be the best near-term solution. In such an approach, the underlying support features of the operating system's TCB would be used to the most feasible extent, and the application would be responsible for additional assurances.

For example, program integrity policies have proven to be an effective measure to prevent users from introducing malicious code into a production system. When coupled with data validity constraints that are subject to its protection, a program integrity policy can provide enough assurance for data integrity in a system-high benign environment.

In a compartmented environment, however, constraint-based integrity coupled with program integrity would not suffice. Integrity policy enforcement in such an environment would have to be based on some type of mandatory integrity policy. Such a mandatory integrity policy would have to have the full cooperation of the TCB to ensure its enforcement. A mandatory integrity policy for such an application obviously cannot be

based on operating system architectural features that are nonexistent. Additionally, the integrity policy in effect must minimize any extensions to the TCB boundary that may be required for its support. Variations on Biba's strict integrity policy may be more appropriate here.

Can a high degree of data integrity coexist with a high degree of trust? The current generation of highly trusted operating systems have not been conclusively examined in this area. Attempts to build high integrity data management applications on Honeywell's Multics system severely limited the user's ability to exercise untrusted applications and other system features. Perhaps if a trusted system were dedicated to a specific application, in execute-only mode, high data integrity could coexist with highly secure operating systems.

CONCLUSIONS

In conclusion, there have been many different integrity policies proposed. Few of them have been tested through a system implementation. Still fewer have actually proven successful in operational environments. Each application must be examined on an individual basis to determine which integrity policy best fits its requirements or if a combination of integrity policies is more appropriate.

REFERENCES

1. Biba, K.J., Integrity Considerations for Secure Computer Systems, ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA, March 1976.
2. Boebert, W.E. & Kain, R.Y., "A Practical Alternative to Hierarchical Integrity Policies", Proceedings of the Eighth National Computer Security Conference, Sept. 1985.
3. Boebert, W.E., et. al., "The Extended Access Matrix Model of Computer Security".
4. Boyun, D., "A New Model of Computer Security with Integrity and Aggregation Considerations", I.P. Sharp Report, 21 March 1978.
5. Boyun, D., "Aspects of Integrity", I.P. Sharp report, March 1986.
6. Coates, C. and Hale, M., ed., Proceedings of the NCSA Invitational Workshop on Database Security, Baltimore, MD., 17-20 June 1986.
7. Committee on Multilevel Data Management Security, Air Force Studies Board, National Research Council, "Multilevel Data Management Security", National Academy Press, 1983.
8. Date, C.J., An Introduction to Database Systems, Vol. I, second ed., Addison-Wesley, 1986.

9. Date, C.J., An Introduction to Database Systems, Vol. II, Addison-Wesley, 1983.
10. Date, C.J., Relational Database, Selected Writings, Addison-Wesley, 1986.
11. Denning, D.E., et al, Secure Distributed Data Views, Security Policy and Interpretation for a Class A1 Multilevel Secure Relational Database System, Interim Report, USAF Rome Air Development Center, Nov. 1986.
12. Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, December 1985.
13. Ferguson, C.T. & Murphy, C.B., "A Proposed Policy for Dynamic Security Lattice Management", Proceedings of the Ninth National Computer Security Conference, Sept. 1986.
14. Fernandez, E.B., Summers, R.C., & Wood, C. Database Security and Integrity, Addison-Wesley, 1981.
15. Landwehr, C., "What Security Levels are for and why Integrity Levels are Unnecessary", NRL Technical Memorandum, 23 Feb. 1982.
16. National Computer Security Center, "Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, DOD 5200.28-STD" (DRAFT), April 1987.
17. Porter, S. & Arnold, T., "On the Integrity Problem", Proceedings of the Eighth National Computer Security Conference, Sept. 1985.
18. Saydjari, O.S., et al, "LOCKing Computers Securely", Proceedings of the Tenth National Computer Security Conference, 23 Sept. 1987.
19. Schell, R.R. & Denning, D., "Integrity in Trusted Database Management Systems", Proceedings of the Ninth National Computer Security Conference, Sept. 1986.
20. Shirley, L.J. & Schell, R.R., "Mechanism Sufficiency Validation by Assignment", Proceedings of the 1981 Symposium on Security and Privacy, April 1981.
21. Ullman, Jeffrey D., Principles of Database Systems, Computer Science Press, Rockville, MD, 1982.
22. Weiderhold, Gio, Database Design, McGraw-Hill, New York, NY, 1983.

STATUS OF TRUSTED DATABASE MANAGEMENT SYSTEM INTERPRETATIONS

Michael W. Hale
NCSC ATTN: C11
9800 Savage Rd.
Fort George G. Meade, MD 20755-6000
(301) 859-4452

ABSTRACT

The National Computer Security Center is developing a document containing interpretations of the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) for database management systems. With each interpreted TCSEC requirement, a rationale for the interpretation is stated. These sections of the document will be supported by appendices that address security issues that are unique to database management systems. The document will be entitled, "Trusted DBMS Interpretations", (TDI).

NCSC COMMITMENT TO DBMS SECURITY

A majority of data processing installations regularly use database management systems. With the expanding information age, the percentage of installations doing data base management is increasing significantly. Unfortunately, the presence of a trusted operating system in these installations does not guarantee that the DBMS can be used to share information in a trusted manner.

There are several characteristics that are common to most DBMS's which make it necessary for them to provide some credible security controls. In many operational environments, users interface directly to the DBMS, which makes the operating system appear transparent. In fact, DBMS designers often choose not to utilize the services provided by operating systems, including the security features. In addition, DBMS's typically provide the capability to share objects that are of a more abstract type than operating systems are capable of recognizing. These characteristics, among others, create the need for security controls within the DBMS to control access to these objects.

Based upon these facts, the NCSC is committed to determining the extent to which database management systems can be trusted to control sharing of sensitive data, and to evaluating and rating commercially available systems against a practical and reasonable criteria.

This commitment will be realized through the TDI that is now being developed. The TDI will strive to specify achievable, practical requirements in order to allow current DBMS installations to become more secure, while laying the framework for

technologically advanced trusted database management systems in the future.

HISTORY AND STATUS OF TDI DEVELOPMENT

NCSC Effort Begins - Spring 1985

In May of 1985 the NCSC began to examine requirements for multilevel data management in an effort to determine whether or not guidance should be written on the subject. It was also part of the task to determine what the scope of such guidance should be. That is, we had to identify what audience could benefit the most from a document on DBMS security. We identified three basic audience segments that could potentially use guidance to be DBMS users, DBMS builders, and DBMS evaluators. Users would benefit most immediately from a document that provided suggestive guidance on how to configure and use existing systems in a secure manner. DBMS builders would need a document that provided TCSEC type criteria on what security features a database management system should have.

Of course the later document would also indirectly help users to improve the security posture of their DBMS installations over the long run, by encouraging the development and evaluation of trusted database management systems. We recognized that this is clearly the most desirable type of DBMS security guideline to be produced by NCSC. We were then faced with the task of determining if it was technically feasible to develop trusted database management systems, which would determine the practicality of a criteria type guideline.

DBMS Security Workshop - June 1986

To determine current technology's ability to support a DBMS security criteria, we organized a workshop to discuss the state of the art in database management system security. The workshop was held in Baltimore during June of 1986 and was attended by 57 experts from DBMS vendors and their customers, government and academia. The participants were divided into three working groups charged respectively with producing reports on security policy, data integrity and inference, and trusted DBMS architectures. Each report details the technologi-

cally possible solutions in each of these areas as well as the problems that require further research and development. These reports and all issue papers written in preparation for the Workshop are available in [CSC86].

Develop Preliminary Drafts - December 1986

At the beginning of FY87, NCSC tasked Mitre and Aerospace to develop preliminary drafts of DBMS evaluation criteria. The drafts were to be in the form of interpretations of the TCSEC for database management systems. This approach was chosen based upon the belief that the TCSEC contains all of the fundamental requirements and control objectives that are necessary for any trusted computer system. In the case of database management systems, as well as networks, the fundamental requirements need to be interpreted more specifically for that type of system.

The preliminary drafts were delivered to NCSC on 31 December 1986. They were used as a baseline to produce the working draft of the TDI.

Working Group Formed to Refine Draft - January 1987

NCSC selected a working group to steer the refinement of the preliminary drafts into a releasable draft. The working group's goal is to produce a releasable draft by late 1987. The group decided at the first meeting to work toward a DBMS document that mirrors the TCSEC in the number of evaluation classes it contains and in the degree of security represented by each class.

While the general feeling is that the evaluation classes in the TDI should be roughly equivalent in features and assurances to the respective classes in the TCSEC, the group did recognize that some requirements which are unique to a DBMS environment will have to be added (e.g., prevention of unauthorized data modification). The group quickly reached a consensus that the requirements must be practical and achievable with current technology.

The TDI will contain appendices that address issues requiring further explanation than can be reasonably provided in the main body of the document. Appendices will address issues concerned with system architecture, including the implications of using multiple hardware bases (e.g., database machines) and the distribution of security functionalities over distinct subsets of the system TCB. Additionally, there will be an appendix that addresses database integrity and consistency.

The group did not attempt to specify requirements for areas that are on the research fringes of DBMS and computer security technology. Problems that are considered to be intractable will be identified and referred to the research and development office of the NCSC. Through

worked examples of trusted database management systems, research and development will be able to prove whether or not pragmatic solutions to these problems exist.

It is expected that research in the area of DBMS security will progress enough over the next five years to enable the resolution of today's research problems. If this happens the TDI will evolve to encompass the newer technology.

SECURITY AT THE SYSTEM LEVEL

The most reliable place to implement security controls on any type of computer system is at the system level. The security controls are much more robust when they are implemented close to the physical representation of the information to be protected. Thus we must require that the system implement as many of the security controls as is technologically feasible to achieve the greatest level of security.

The main body of the TDI will specify requirements that DBMS vendors provide security mechanisms to control the sharing of databases. The TDI will not enable the evaluation of database management systems in isolation. Instead, it will require that they be evaluated in the context of their supporting operating system and hardware base. Thus the evaluation will be of specifically configured systems, which are the most robust way to ensure that the various security mechanisms work together properly.

TCB SUBSETS AND INCREMENTAL EVALUATION

A relatively new concept addressed in the TDI is that of TCB subsets. TCB subsets occur when the DBMS relies upon the operating system to provide it with a portion of the overall system's security features and assurances. The TDI will specify prescriptive requirements for how TCB subsets must interface and function as a complete security system. An in-depth discussion of this concept will be provided in an appendix to the document.

The precise way in which the TCB subsets must interface is that they must be implemented in layers, where the security policy of a lower level subset is used by all higher level subsets. The higher level subsets can never bypass the security policy of the lower levels, but they may add their own security policy that does not conflict with the policy of the lower levels.

If TCB subsets interface in a precisely defined way, an evaluation methodology that will become known as incremental evaluation will be used to evaluate the DBMS. This will be possible when a DBMS is undergoing an evaluation in the context of an operating system that has been previously evaluated, or is being evaluated at the same time. If the DBMS uses the security policy of the underlying operating system, and is layered properly on top of the operating system, then this system is a candidate for an

incremental evaluation that can be done in two increments. The results of the operating system's evaluation can be used as input to the evaluation of the operating system-DBMS combination. Note that in the event that the DBMS bypasses any operating system services that were included in the previous evaluation, the incremental evaluation methodology cannot be used, as the base TCB subset has been altered.

The rating resulting from the incremental evaluation would apply only to the aggregate of the TCB subsets when used together. It would not apply to the DBMS TCB subset if it were ported to another operating system. The rating of the overall TCB must always be less than or equal to the ratings of all of the previous increments in the evaluation. That is, adding a TCB subset cannot improve the rating of the base TCB.

If the additional DBMS TCB does not interface correctly with the base TCB, the incremental evaluation methodology cannot be used. For example, performance considerations might dictate that the DBMS be designed to bypass some of the operating system's services. In this case, the entire system must be reevaluated in its entirety, because the TCB that was present in the operating system is not being used by the DBMS. This type of evaluation methodology is directly analogous to a traditional operating system evaluation. Since the original operating system rating is invalid, it is possible that the DBMS evaluation could result in a higher rating than the operating system had received.

EXPLICIT REQUIREMENTS FOR INTEGRITY

The TDI will place significantly more emphasis on integrity of the protected data than the TCSEC does. The concept of integrity is divided into two fundamental areas: unauthorized modification of the

data, and consistency and correctness of the database. The requirements for unauthorized modification will be integrated into the security policy of the system. Separate requirements might be written to ensure that the DBMS contains features to ensure the consistency and correctness of the database. However, as of this writing, this area is less understood, and it is questionable as to what level of assurance we can get that consistency and correctness controls are correct. As a matter of fact, there is not unanimous agreement that consistency and correctness requirements belong in the TDI, as some view them as DBMS operational requirements as opposed to DBMS security requirements.

ACKNOWLEDGMENTS

Credit for many of the ideas in this paper is due to the members of the Trusted DBMS Interpretations Working Group: Dr. D. Elliott Bell, Dr. John Campbell, Dr. Deborah Downs, Kenneth Eggers, Richard Graubart, Neal Haley, Ronda Henning, Terry Mayfield, Dr. Robert Morris, Dr. Roger R. Schell, Dr. T.C. Ting, Mario Tinto, and Grant Wagner. I express my appreciation for the ideas and insights that these people have given to the Center's efforts in DBMS security.

REFERENCES

- [CSC85] National Computer Security Center, Department of Defense Trusted Computer System Evaluation Criteria, August 1985.
- [CSC86] National Computer Security Center, Proceedings of the National Computer Security Center Invitational Workshop on Database Security, Baltimore, Maryland, 17-20 June 1986.

Insider Threat Identification Systems

Allan R. Clyde

A. R. Clyde Associates
10101 Grosvenor Place, #2006
Rockville, MD 20852

Abstract

Motivation is established for the importance of addressing the risks arising from insider threat on automated information systems. The insider threat is characterized and the types of damage to the system sponsor are outlined. The concept of an insider threat identification system is introduced as a framework and a discipline for addressing this threat. The basic components of such a system are outlined. Mandatory, internal system surveillance is identified as the foundation component of an insider threat identification system. Its characteristics are discussed and the current status of surveillance technology is noted. The second component is a capability for analyzing the data captured by system surveillance. The work done in this field is reviewed. An expert system for analysis of a surveillance knowledge base to identify suspicious events is proposed. The last three components of an insider threat identification system are outlined. Their dependence on a keystroke and system response level of surveillance is noted. However, discussion of these components, dealing with investigative evidence gathering, damage assessment and recovery support, are outside the scope of this paper. This work has been privately funded in the interest of product and market development.

1. Introduction

It is widely recognized among the many critically concerned professionals and policy makers in the Infosec community that managing the risks arising from insiders on sensitive computer systems is of major and growing importance.¹ An insider may be characterized as a member of a population of trusted users for

¹ A number of policy makers and professionals have gone on record about these concerns and the importance of managing the risk from insider threat. The following are excerpts from some open letters to the author: "We appreciate your effort in addressing new technologies on the insider threat against sensitive computer systems. We are also pleased to note that you are working closely with the National Computer Security Center and its new Chief Scientist in this regard. . . . We encourage you to continue your valuable work to provide computer security products, especially those designed to specifically counter the insider threat, . . ."—Donald C. Latham, Assistant Secretary of Defense (C³I), January 2, 1987. "It is my firmly held opinion that a substantial majority of financial losses suffered in the private sector are caused by insiders. . . . In most cases, the acts that have caused the losses have been done by persons who had the authority to perform the acts. . . . Any contribution to the interpretation of audit information (computerized or otherwise) to counter these threats would clearly advance the interests of computer security,"—Dr. Robert Morris, Chief Scientist, National Computer Security Center, November 20, 1986. "I sincerely appreciate your bringing these [insider threat] concerns to my attention and your continuing interest in developing technologies to counter the insider threat, . . ."—Harold E. Daniels, Deputy Director for Information Security, National Security Agency, May 4, 1987. Late in November, 1986, the Associated Press reported that the President had sent an anti-spy master plan to the House and Senate intelligence committees. This news item stated in part: "The president's plan is an unprecedented blueprint for broad based reform to U.S. efforts to counter the Soviet bloc intelligence threat. . . . The Defense Department is directed . . . to provide monetary or administrative penalties for contractors with security lapses and bonuses for those with tight programs. . . . Additional research is promised on technical safeguards for secrets stored in computers. . . . sooner or later we'll come across a spy case involving computer theft of secrets."

a given installation; that is, an insider has authorized access to the automated information system and the resource it manages. Access limits for each insider are set by security policy and enforced by the computer access controls. The degree of trust placed in a given insider may vary from one site population to another, and it may also vary within a population. The degree of trust assigned to each insider is generally a consequence of a formal review or investigation of the person's background and integrity.

Inappropriate conduct of an insider in the use of an automated information system constitutes a threat of potential damage to the sponsor of the system. This conduct may be characterized as any use of the system that violates actual or intended policy. Unauthorized persons who by skill may penetrate the access controls of the system are regarded as *intruders*. Such persons have the same potential for damage to the sponsor as does the inappropriate conduct of insiders. Where such insiders may also seek to extend the boundaries of their authorization, the distinction between these insiders and intruders is not important relative to the technologies required for the protection of the sponsor. The sponsor may be an agency of the government, a military group, a government contractor, a national laboratory, or an entity in the private sector.

Damage to the sponsor may take a number of forms. Much has been reported, and the following is a distillation of the basic categories:²

Denial of Service: The system becomes inoperable, unresponsive or corrupted in some manner. Some or all of the users are unable to perform their tasks on the system in a normal way.

Information Loss: Information managed by the system is lost by destruction or corruption.

Disinformation: Information is altered in a manner that misleads.

Information Compromise: Information is conveyed to persons not authorized by policy to receive it.

Resource Exploitation: The system is used to promote objectives within or outside of the system that are not authorized by policy.

Damaging conduct in the use of automated information systems may be a consequence of ignorance, error or malfeasance.³ This paper addresses the identification of insider threat arising from any conduct that damages the sponsor by violating actual or intended policy. The objectives of such a system include the identification of the perpetrator and assessment of the damage, together with support procedures for recovery. These objectives depend on the successful collection and analysis of detailed surveillance data. When these objectives are served, it is also possible to take corrective measures that increase the

² This categorization of damage has been distilled from a wide range of recent publications discussing and reporting computer crime, including technical journals, trade magazines and the public media. These summarized categories appear to include the various forms of damage being currently reported by the sponsors of automated information systems.

³ From the information reported in the press, the spy case involving Jonathan J. Pollard included the theft and compromise of large volumes of information managed by a secure, automated information system at a military site. This site was protected by computer access controls supplied by a major government vendor. These widely used access controls were not effective against insider threat. Similarly, such access control also fails to offer protection to the sponsor from the damaging potential consequences of ongoing training inadequacies and human error.

resistance of the system to the kind of threat encountered. Security policy can thus be updated, and the administration of access controls can subsequently be improved.

Damage to the sponsor may occur entirely within the framework of authorized access to the various objects protected within an automated information system. Examples of such objects are files, programs and memory structures. It is clear that the access controls of a trusted computing system will not provide protection under these circumstances. However, insiders perpetrating damaging activity are being detected at a number of sites with user-interaction surveillance technology.⁴

Compromise and exploitation may also occur through a penetration of access controls and an expansion of access boundaries by skilled insiders. Instructions on how to penetrate the secure, general purpose computer systems now in wide use may often be found on the college campus and in the university library.⁵ There are only a few systems that have been, or soon will be, rated by the National Computer Security Center as having access controls that are trusted at the high end, that is, at the A1 level of security.⁶ The trusted systems now in dominant use have ratings at the C1 and C2 levels and are not very tamper-resistant. For example, trojan horse attacks and the insertion of viruses and worms continue to be quite successful [18]. Even should access control technologies, together with their availability and cost, meet the community's highest expectations, technologies that identify insider threat remain a complementary and essential part of computer security.

Where many of the critical systems of interest are physically secured within vault-type buildings, it is clear that the management of risk arising from insiders is of dominant importance in protecting sensitive, automated information from compromise and exploitation. Historically, the management of this risk has been sought by physical security, computer access controls, and a suitable clearance for the granting of trust to users. However, as evidenced by the Jonathan J. Pollard case and similar cases, technology that identifies insider threat needs to address the problem of altered motivations in populations of trusted users. The same technology can also address

⁴ A number of cases have been reported to the vendor of the SentryGATE product concerning the detection, evidence gathering and termination of damaging activities by insiders on secure systems through the deployment of surveillance technology. The access controls supported by a trusted computing base are not intended to address the detection of insider threat, but rather, the prevention of unauthorized access [1]. Case studies of perpetrator identification, evidence gathering, damage assessment and recovery are in preparation at a number of sites.

⁵ It has been the habit of the vendors of the most widely used secure systems to publish lists and descriptions of problems fixed in new versions of these operating systems, where such systems either contain or support the trusted computing base. These systems include MVS, VMS and UNIX. This information has traditionally been available with operating system documentation and its updates. It may therefore be found, unrestricted, on college and university campuses, in association with those computer systems and the various related libraries. A perpetrator may often be able to depend on a given installation to be slow in upgrading to the new version. In this case, the subject documentation becomes a textbook on system penetration.

⁶ The National Computer Security Center has been successful in completing an evaluation of the Honeywell SCOMP system at the A1 level. Others are expected in the future. However, complexities of implementation and of suitable evaluation for such systems are considerable. These lead to high costs and substantial delays in an environment of limited resource. The Center publishes an Evaluated Product List, as a service to the community, where information can be found on each product that has achieved evaluated status.

the risks that arise from inadequate training, human error, misdirection, or loss of suitable accountability and supervision, as in the Oliver North case.⁷

The historical perspective on trust and the notable lack of technology that can identify insider threat have led to widespread, blind trust of user populations.⁸

There is also a pattern of exemptions to policy guidelines in those areas that require a surveillance-oriented technology.⁹ This circumstance may no longer be necessary. It is in this context, and in the backdrop of similar expressions of concern found throughout the Infosec community,¹⁰ that this paper introduces and outlines some proposed characteristics of insider threat identification systems. The specific types of concerns expressed throughout the Infosec community about managing the risks from insider threat suggest a system that consists of at least these five basic components:

- (1) Capture of detailed system and user monitoring data through high performance surveillance technology
- (2) Expert system analysis of a surveillance knowledge base for suspicious events, with weighted scoring
- (3) Identification of perpetrators by an interactive expert system, using the analysis results and the

⁷ The report by the Tower Commission, acting under presidential direction, presents evidence of directives made by electronic mail on sensitive computer systems at the National Security Council that demonstrate a loss of suitable accountability and a use of government computers to further objectives alleged to be contrary to the sponsor's policies and objectives.

⁸ Five groups are known to the author to have published in the area of insider threat identification. Two of these, the groups at SRI International [3 and 13], and at the National Computer Security Center [2], are characterized by their titles as intrusion detection. The Sytek, Inc. group speaks of analyzing traditional, existing audit trails for security violations [4]. On page 71 of this reference, under the section called Background, it states:

"Monitoring of computer system use for security violations will always be necessary. Even if we perfect the ability to design secure computer systems which we can trust, we can never fully trust their users. The problem of catching legitimate users who violate system security will remain a problem which can most effectively be addressed by security monitoring.

"Currently, system security officers perform security monitoring of computer systems by manually reviewing the system audit trail. The only automated help available to them comes in the form of audit mechanisms capable of producing reports or data bases which store audit trail data. Consequently, there is a great need for more capable automatic tools to assist in this task. This need, and the lack of work being done to develop such tools, was pointed out by Marv Schaefer in his closing remarks to the Eighth National Computer Security Conference. Although in 1980, the James P. Anderson Co. produced an excellent discussion of this problem, not much seems to have been done since then." (see reference [17]).

The fifth group, A. R. Clyde Associates, with collaborators at Clyde Digital Systems, has addressed insider threat identification systems as a whole. Most of the Clyde Digital Systems work has been done on surveillance technology as a foundation for such systems. However, the SentryGATE product now in the field also performs surveillance data analysis using a small number of suspicious event tests with weighted scoring.

⁹ The author has received reports of a number of instances where exemptions to government guidelines for monitoring the activities on critically sensitive systems have been granted, in the belief that no effective technology exists in support of such guidelines. In particular, a classified document produced by the Defense Intelligence Agency, called DIAM 50-4, is known to contain such guidelines.

¹⁰ The author refers here to meetings and conversations with members of the National Telecommunications and Information Systems Security Committee, staff personnel at NSC and NSA, SAISS committee members, National Computer Security Center chiefs and others in the government and military intelligence community. There has been a unanimous expression of concern about insider threat on automated information systems and recent reports of serious compromise to the national interest from same.

- surveillance data set for investigation, evidence gathering and case development
- (4) Tools for a detailed damage assessment from the surveillance data set
 - (5) Support capability for recovery, based on successful damage assessment and the surveillance data set

2. Internal Surveillance of Automated Information Systems

Methods of internal surveillance in an automated information system fall into two categories, depending on objectives. The first is passive detection, which means that the in-depth analysis of surveillance data is performed for the detection of insider threat takes place off-line or off-hour. The second may be characterized as pro-active detection, which means that the surveillance data are tested in real time for certain events that the tests lead, when necessary, to an immediate protective response. The objective of pro-active detection is to recognize and suspend processes that may cause serious, immediate damage to the system. The objective of subsequent in-depth analysis of the surveillance data is effective detection of the more subtle threats that are, in a broader sense, damaging to the sponsor.

2.1 Full-system Surveillance

For convenience in subsequent references, the term *full-system surveillance* refers to the capturing and recording of all surveillance data that appear to have an important bearing on the successful detection of insider threat. Such full-system surveillance falls naturally into three independent functional modules, referred to here as: 1) user-interaction, 2) batch stream, and 3) system-event [6].

2.1.1 User-interaction Surveillance

User-interaction surveillance must deal with both local and remote terminal users. These are dialup users, or users accessing the system from another node on a network. This module is responsible for monitoring the information that moves between the processor and the terminal, including undisplayed information such as escape codes and control codes. The security policy at some installations requires the ability to play back a terminal session from the surveillance data exactly as it originally occurred.¹¹

2.1.2 Batch Stream Surveillance

Batch stream surveillance records the contents of batch and command control files and the system's responses to these files. This module is responsible for monitoring the information that moves between a batch stream and the processor. The security policy at some sites may require the ability to play back a batch stream activity at a terminal as though it had been performed manually, showing the command input and the system responses.

¹¹ A number of sites, where surveillance technology is now integrated into security policy and procedures, are performing *blanket* monitoring. This is a continuous, mandatory capture of all keystrokes and system responses for each user. Passwords are excluded.

2.1.3 System-event Surveillance

System-event surveillance includes file I/O activity and system services. These surveillance data characterize the response of the system to users and processes. All I/O calls and system service calls may need to be monitored and the corresponding surveillance data captured, depending on the requirements imposed by security policy. Some installations may require total system surveillance and may require that data be captured in extensive detail.

It is recommended that the system-event surveillance module also collect independent system accounting statistics. Contributions to the full-system surveillance data set should not depend on an extraction from records handled and managed by the operating system and systems management personnel. Independence of data source and layered tamper-resistance are important to assure data integrity, even if the trusted computing base should be penetrated.

2.2 Subsequent Detection

The analysis of full-system surveillance data for certain kinds of suspicious events also requires correlation with corresponding data from prior periods. The limited experience with analysis obtained to date¹² indicates that the more subtle forms of computer crime are patterns of conduct that unfold over a period of time. Detection is greatly improved by analysis of historical use patterns. Passive detection is characterized by the collection of substantial amounts of data for off-line or off-hour analysis, which does not burden the system in performing its intended function.

2.3 Pro-active Detection

Pro-active detection must consist of a carefully limited set of tests in order to avoid burdening the system. These tests are performed against the surveillance data in real time, that is, as the data are being collected. The amount of testing is guided by security policy, the potential system burden, and the necessity of preventing system-damaging events. Upon detection of a suspicious event, an alarm is sent to the system security administrator and the suspicious task is suspended, pending a more in-depth review of the potentially damaging activity. The task may subsequently be terminated or re-activated, depending on the decision of the system security administrator. For some tests, additional computer analysis may be useful following detection of a suspicious event. Such analysis could be automatic, or it could require interactive, investigative intervention by the security administrator.

2.3.1 Dealing With Trojan Horses, Viruses and Worms

Should security require that each program in the system library include a table that lists the resources needed for its execution, then a system-events surveillance module could deal pro-actively with clandestine attempts to insert damaging logic into the system. Such a table could be interrogated prior to

¹² In the case of Jonathan J. Pollard, suspicion was finally raised through an observation by personnel that he seemed to be accessing large volumes of data. As an authorized user, access controls offered no protection to the sponsor, nor would the analysis of any given day's activities necessarily have shown a sufficient pattern for suspicion. However, had an insider threat system been employed to analyze surveillance data using present and prior activities matched against a user profile table, it seems likely that such suspicion would have been raised at greatly reduced levels of compromise.

execution. The resources subsequently requested by the program could then be tested for compliance with the resources authorized in the table. When lack of compliance is detected, the program could be suspended. Appropriate alarms could be sounded and interactive, investigative procedures initiated.

The resource table could be encrypted. Access to the decryption key could be limited to the system-events surveillance module. In those cases where fully automatic code generating tools are used to create system library programs, the table would be constructed and encrypted by computer. The program output from such a code generator could go by trusted path directly to write-once media. This arrangement could virtually preclude human interaction with code placed into a certified standard master library. The programs in the system library would be copied from this independently secured master library and then periodically bit-checked against it for correctness.

2.4 Attended, One-on-one Surveillance

In addition to the unattended, pro-active detection just suggested, attended, one-on-one surveillance of certain users by the system security administrator has been found useful as an investigative tool.¹³ The technology referenced here has been in the field for about eight years. It is characterized by high performance, in that it does not place a burden on the processor. The investigator or security administrator may interact directly with the process when necessary.

2.5 The System Relationship of Insider Threat Components

The surveillance component of an insider threat identification system is the only one of the five components that requires a close working relationship with the operating system and its trusted computing base.¹⁴ The other four components may run on-line or off-line as trusted application software.¹⁵

¹³ There is a product in the field that operates at a subject-object, primitive-pair level, employing an S-gate technology. A discussion of these concepts is found in reference [6]. A technical overview of this product, called CONTRL, can be obtained from Clyde Digital Systems. This technology has been used in concert with system surveillance technology in criminal investigations on automated information systems. It allows an investigator to dynamically link to an arbitrary given terminal and surveil all information moving between that terminal and the processor. The perpetrator is unable to detect the presence of this one-on-one surveillance activity. The session is recorded and the results can be compared for added perfection of evidence with the raw surveillance data set.

¹⁴ The operating system or the trusted computing base would contain the instructions for transfer of program execution, at appropriate times, to the surveillance modules. In order for these branching instructions to be inserted dynamically into the trusted system without interfering in any way with its operation and certification, the logic for inserting the surveillance modules depends on a certain, precise knowledge of the subject target system. The target system need not have any particular accommodation in anticipation of, or in an attempt to support, the surveillance modules outside of its normal, certified function. This broadly characterizes the relationship between the trusted computing base and the surveillance technology now in the field.

¹⁵ The remaining four components of an insider threat identification system, namely expert system analysis, interactive investigation, damage assessment and recovery-support, need not impact the certification of the target system under surveillance in any way. They may be executed as trusted application software in an off-hour mode. In some cases it may be appropriate to place these components on a system dedicated to insider threat identification in support of a number of target systems under surveillance.

2.6 Surveillance Issues for Optimizing Detection

In order to optimize the ability to detect suspicious events through analysis of the surveillance data set, it is essential that policy dictate mandatory surveillance.¹⁶ To achieve mandatory surveillance, the technology must be tamper-resistant, must not burden the processor and must be a cost-effective adjunct to a certified trusted computing base. This is necessary in order to enforce such a policy and to make it viable.

Recent experience with a criminal penetration of a secure system has emphasized the need for correct operation of the surveillance component, even when the access controls of the trusted computing base are penetrated and its alarms and auditing turned off.¹⁷ This level of tamper-resistance is best assured when there are no operating system services that make reference to, identify, or grant access to the surveillance component. This is a degree of concealment. Such concealment, or "data hiding,"¹⁸ has proven itself in the field by delivering

¹⁶ Discretionary surveillance may be used to gather evidence upon established suspicion. However, permitting discretion reduces the tamper-resistance of the surveillance function by granting discretionary access at some level of authorization for enabling and thus for disabling surveillance. In addition, the chain of evidence to the initial compromise will almost always be lost; the perpetrator may escape notice and the innocent may come under suspicion. Incomplete surveillance data limits the ability of expert system analysis for suspicious events. In most cases, traditional audit technology is regarded as too limited in detail and too burdensome to the target system to be used in a manner sufficient to the demands of optimum expert system analysis. In some cases, suspicion can only be established at a prompt level for certain dangerously privileged programs.

¹⁷ In an investigation by the FBI on a system for which the author had responsibility, an intruder was observed to penetrate the access controls of a secure operating system. The intrusion was first noted by a system security administrator when scanning the job queue during the course of a normal procedure. A suspicious activity was noted and the security officer executed one-on-one surveillance against the remote terminal under suspicion. The system had a monitoring policy which was being enforced by mandatory, blanket surveillance. This surveillance also recorded the activities of the security officer in observing the perpetrator directly. The perpetrator was observed to penetrate all levels of system privilege and then to disable the alarms and auditing of the trusted computing base. Certain features of the penetration pointed clearly to insider knowledge of the system and, therefore, a knowledge of the existence of the surveillance function. In over 25 hours of recorded activity on the system, the perpetrator committed criminal acts, but did not disable the surveillance function. A complete chain of evidence to the initial compromise was located and extracted from the raw surveillance data set.

¹⁸ On page 49, section 4.1.3.1.1 of the Criteria [1] on the subject of class A1 operational assurance, the National Computer Security Center states: "The TCB shall incorporate significant use of layering, abstraction and data hiding. . . ." In the same context it states: "The TCB shall maintain process isolation through the provision of distinct address spaces under its control. . . ." It also states: "The TCB shall maintain a domain for its own execution that protects it from external interference or tampering. . . ." These concepts should also be applied independently to the design and implementation of surveillance modules. The user-interaction surveillance technology now in the field utilizes each of tamper-resisting mechanisms in its relationship to the TCB. Therefore, should a penetration of the TCB occur, an independent layer of logic maintaining process isolation with control over its own distinct address space can retain its integrity and monitor the penetration. Footnote 17 describes such an instance. In addition, it is proposed that the branching locations dynamically inserted into the TCB for transfer of process execution to the surveillance modules be restricted information. This adds an additional increment of tamper-resistance. It should be noted here that the alteration of a given branching location in the TCB by a perpetrator will not likely disable certain advanced surveillance technology currently in the field. Because of the independence of the surveillance modules and their capacity for dynamic insertion, restricting the information on the branching locations does not impact the public-knowledge practices of key operating system vendors for source code on C1 and C2 rated systems.

mandatory, user-interaction surveillance that cannot be interrupted by any particular privileged use of the trusted computing base or operating system.¹⁹ The surveillance data set can also be made more tamper-resistant through the use of existing write-once media, such as optical storage, and through suitable encryption.

2.7 Surveillance Field Experience

Field experience with existing user-interaction surveillance technology has also been particularly encouraging for its high performance. When *blanket* monitoring (the continuous capture of all keystrokes and system responses at the terminal for each user, excluding passwords) of all users is applied, which is the mode required by security policy at a number of sites, the user-interaction surveillance technology does not reveal itself to the user community as an observable burden on the system. Measurements to date of processor time required for the execution of this surveillance function consistently lie below 3%.²⁰ This experience, together with recent development work in the system-event area, indicates an expectation of high performance for total system surveillance, which would include all three classes of surveillance and the capturing of data at a detailed level for all users.

2.8 Cost Effectiveness

Cost effectiveness in surveillance technology is essential to the viability of a mandatory, full-system surveillance policy. The cost components involved in making effective use of surveillance technology on existing systems have been relatively modest. These components, from experience in the field with user-interaction surveillance, are limited to acquisition, integration and training. The acquisition costs have been particularly modest. Integration involves only updates and extensions to security procedures. There are no changes to the certified trusted computing base. The user-interaction surveillance technology can be dynamically inserted into a system in normal operation without interfering with its current processing in any way. It is this technique, together with concealment,²¹ that precludes the

¹⁹ The surveillance technology now in the field is not accessible by services provided by the operating system or Trusted Computing Base with which it is installed.

²⁰ The measurement data on user-interaction surveillance is limited. However, the tests run with instrumentation in the code for processor-use measurements consistently indicate a utilization of less than 3% for the surveillance system in relationship to all other activities over various sample increments of time. The tests were made for the blanket monitoring mode, which is the maximum system loading condition. The job-mix was typical of a highly interactive environment.

²¹ The requirement of a correct, continued execution of the operating system and the trusted computing base, together with the tasks in execution, under dynamic insertion of surveillance technology is a valuable form of measurable assurance of correct operation, functional independence and domain isolation of each. On this subject, and relative to the operational assurance of a trusted computing base for class A1, the National Computer Security Center states the following on page 49, section 4.1.3.1.1 of the Criteria [1]:

"The TCB shall maintain a domain for its own execution that protects it from external interference or tampering. . . . The TCB shall maintain process isolation through the provision of distinct address spaces under its control. The TCB shall be internally structured into well-defined largely independent modules."

In addition, by precluding access to the surveillance technology through the services of the trusted computing base and through the operating system, the certification of these components is not effected by the presence of the complementary surveillance functions as it supports the identification of insider threat. This form of concealment therefore has an important bearing on the question of re-certification, and its costs, where access controls and surveillance are both required by policy.

need for re-certification of the trusted computing base and the high costs for same.

This is the primary contributor to the excellent cost effectiveness of user-interaction surveillance in the field today. It is copied on and executed. It runs unattended and uninterrupted. This favorable experience suggests the importance of retaining the capability of dynamic insertion and concealment in the implementation of the system-event and batch stream modules, which, together with the existing user-interaction modules, would make up full-system surveillance.

3. Suspicious Event Analysis

The analysis of data gathered through the internal surveillance of automated information systems appears to have only recently been addressed by technical professionals in the Infosec community. There appear to be just five groups that have published papers in the field. In addition, one of the intelligence agencies is known to have implemented suspicious events tests for the UNIX environment and has had such tests in operation for a few years. Others may be practicing some audit trail analysis.²² The UNIX operating system provides a number of user- and system-activity audit trails of limited detail [14] that may, nevertheless, be used to detect inappropriate conduct with some degree of success. However, such data lacks the detail necessary to investigate the subtleties of many forms of damaging conduct by authorized users.²³

3.1 The Intrusion-detection Expert System

A paper by Dorothy E. Denning of SRI International, one of the five groups, appeared in early 1986, based on work reported internally in 1985 [13]. This paper proposes a model for a real-time intrusion-detection expert system (IDES). "The model is based on the hypothesis that exploitation . . . [will involve] an abnormal use of the system" [3]. Denning points out four factors that motivate the development of such a system:

- (1) Most existing systems have security flaws that render them susceptible to intrusions, penetrations, and other forms of abuse; finding and fixing all these deficiencies is not feasible for technical and economic reasons;
- (2) Existing systems with known flaws are not easily replaced by systems that are more secure—mainly because the systems have attractive features that are missing in the more-secure systems, or else they cannot be replaced for economic reasons;
- (3) Developing systems that are absolutely secure is extremely difficult, if not generally impossible;
- (4) Even the most secure systems are vulnerable to abuses by insiders who misuse their privileges. [3]

²² It is likely that some manual, and to some degree automated, analysis of certain traditional audit trail data has been practiced in recent years by various government, military, contractor and private sector groups. However, no formalization of an insider threat identification system discipline appears to have developed.

²³ For example, the operational support of an automated information system by security administrative and system management personnel requires the use of dangerously privileged programs. Such programs include those that are able to alter passwords, set up accounts, change levels of access, install privileged programs, disable alarms, alter traditional audit mechanisms, and alter the operating system or trusted computing base. In the case of what appears to be an authorized use of such programs, suspicious activity can best be assessed at the prompt level. An effective investigation that confirms suspicion or establishes innocence also depends critically on evidence gathered at the keystroke and system response level.

These same factors represent concerns throughout the community relative to insiders engaged in conduct that violates security policy. Such persons may also engage in intrusive behavior through ignorance, error or malfeasance.

In the IDES approach [3], a *subject* or group of *subjects* is characterized by an activity profile with respect to the *objects* normally accessed.²⁴ The audit record data of the system are tested to detect abnormal usage by matching them statistically against the activity profiles. The ability to update the activity profiles and to change the pattern matching rules implies an expert system where a portion of the surveillance knowledge base may, in a statistical sense, learn from individual use patterns over a period of time.

The value of these techniques can be extended to off-line or off-hour analysis, including cross-correlations of use patterns and their corresponding activity profiles with prior period data. This and other suspicious event tests, which are amenable to subsequent in-depth analysis, are further enhanced by the greater detail of surveillance data. For example, it is possible to support, at a keystroke level, activity profiles for certain users who execute dangerously privileged programs. Such detailed data will permit the detection of an anomalous or peculiar use of selected interactive programs. This is also the case for critical system services and batch command sequences.

3.2 Detection By Pre-assessed Intrusion Patterns

Other work on intrusion-detection, related somewhat to the SRI International activity [3 and 13], was reported at the 9th Annual National Computer Security Conference in September 1986 by Lawrence R. Halme and John Van Horne of Sytek, Inc. [4, 10 and 16]. This work was based on a fixed set of pre-assessed intrusion patterns. Functions of certain fields found in traditional audit records form what they have called *features*. Parameters in these features are set to values found to characterize normal user patterns *learned* over some number of sessions of a given kind. These features are then used to discriminate between normal and intrusive behavior. Successful features are combined to create an activity profile for each user. It seems intuitive that higher resolution in the data tested, that is, greater session detail, would improve the discrimination possible by this technique. In particular, certain tested features were reported to have failed for lack of data of sufficient quality. In conclusion, the paper states in part: "It is also important to determine what other monitoring data, not normally contained in audit trails, would be useful" [4]. More work on generic feature development, based on detailed system surveillance data, is indicated by the successful features reported.

²⁴ A *subject* may be characterized as a person or a program which acts upon an *object* within the automated information system. An *object* may be characterized as an item of information, such as a record, a file, a system table or a memory structure. It may also be a program. The access of a given object by a given subject is mediated by a *security kernel* which implements a *reference monitor* to validate each access. The reference monitor was introduced in a report by James P. Anderson Co. in 1972 [12]. This report describes the concept of "a reference monitor which enforces the authorized access relationships between subjects and objects of a system." A *reference validation mechanism* was further defined as, "an implementation of the reference monitor concept... that validates each reference to data or programs by any user (program) against a list of authorized types of reference for that user." The Criteria of that National Computer Security Center defines a security kernel as: "The hardware, firmware, and software elements of a trusted computing base that implement the reference monitor concept. It must mediate all accesses, be protected from modification, and be verifiable as correct."

3.3 Intrusion Detection by Analysis of System Service Calls

The work by Jeffrey D. Kuhn of the National Computer Security Center was also reported in the 9th Annual National Computer Security Conference [2]. This work is based on an analysis of those system service calls that are useful in detecting intrusion, as recorded by traditional audit trail techniques. In the conclusion of the paper, Kuhn reports that "an examination of operating system penetration techniques and current auditing methods indicates that most sophisticated violations of system security will be completely undetected, leaving potentially no trace in the audit logs at all" [2]. This is disappointing, but not unexpected.²⁵ It is recommended that system-event surveillance data be correlated with user-interaction and batch stream surveillance data. As Kuhn points out [2], these data should be gathered at the lowest possible levels.

If the data are not gathered by a mandatory mechanism that is secure from tampering even when the trusted computing base is penetrated and all privileges compromised, then any analysis of those data is based on potentially limited or lost information and on possible disinformation.²⁶ Nevertheless, a variety of suspicious event tests, together with techniques for evaluating use patterns, have had some success in detecting conduct that violates security policy. Correlated data on system-service calls can only enhance that success by extending the tests that can be performed and by ratifying the expected relationship between system-service activities and user-interaction or batch stream activities. Alternatively, such data could be used to disclose an unexpected relationship, indicating a particularly subtle and sophisticated abuse of the system and its trusted computing base.

The author has proposed the collection of full-system surveillance data by an *S-gate* technology. This technology mediates and surveils a common data path between *subject-object primitive pairs* [6]. This is the technology that was used to implement the existing user-interaction surveillance system now in the field.²⁷

²⁵ Footnote 17 describes an investigation into an instance of criminal conduct on a secure system where the perpetrator left no evidence in the form of audit logs or alarms. These mechanisms were turned off by the perpetrator before proceeding with the crime.

²⁶ Relative to these concerns, Lawrence R. Halme and John Van Horne state [4]:

"An automatic tool to assist in the task of security monitoring would require data about user activity on the system. Audit trails already provided by the system are one source of such data. They have the advantage that they are an economical and practical source, since their use would require the automatic monitoring tool only to interpret the data and not collect it. On the other hand, the disadvantage of the use of audit trails should be recognized. They may not have been originally intended for security purposes and may not contain enough security relevant material. The audit mechanism may not be secure itself, so that the audit data it produces may be of questionable integrity."

²⁷ In reference [6] on pages 1 and 2 the reader will find a discussion of *S-gates*, and again on pages 11 and 12. For simplicity here, the author has combined the local user-interaction *S-gate* and the remote-terminal *S-gate* into the single surveillance model, called herein the user-interaction surveillance module. In reality, this capability is represented by more than one software module with independent domains in the existing implementation. Similarly, the system-call *S-gate* and I/O-call *S-gate* are combined in what has been termed the system-event surveillance module for simplicity in discussion. The history of *S-gate* development is presented on page 3. Robert A. Clyde is recognized for the design and implementation of the first, and successful, *S-gate*, with the original concepts going back to 1975. This work was done at Clyde Digital Systems, under the direction of the author and with private funds. Also, a discussion of *subject-object primitive pairs* may be found on pages 11 and 12.

3.4 Computer Security Threat Monitoring and Surveillance

A report entitled "Computer Security Threat Monitoring and Surveillance" was written under contract 78F296400 by James P. Anderson Co. in early 1980 [17]. The study was performed for "the purpose [of improving]... the computer security auditing and surveillance capability of the customer's systems" (Section 1.1, Introduction [17]). For background (Section 1.2), the report states in part:

Audit trails are taken by the customer on a relatively long term (weekly or monthly) basis. This data is accumulated in conjunction with normal systems accounting programs. The audit data is derived from SMF records collected daily from all machines in the main and Special Center. The data is temporarily consolidated into a single file... from which the various summary accounting and audit trail reports are produced. After the various reports are generated, the entire daily collection of data is transferred to tape. Several years of raw accounting data from all systems are kept in this medium.

Audit trail data is distributed to a variety of individuals for review... [This includes] activity security officers for some applications located under their purview, but the majority [goes] to the customer's data processing personnel. For the most part the users and sponsors of a data base of an application are not the recipients of security audit trail data. [17]

The term SMF means system management facilities and refers to traditional audit trail information collected on IBM mainframes, such as session, time, resource use, user identification, programs run, files opened, reads, writes and related statistics. This section of the report goes on to characterize the value and the limitations of this class of raw surveillance data:

Security audit trails can play an important role in the security program for a computer system. As they are presently structured, [these data] are useful primarily in detecting unauthorized access to files. The currently collected customer audit trails are designed to detect unauthorized access to a dataset by user identifiers. However, it is evident that such audit trails are not complete. Users... with direct programming access to datasets... may operate at a level of control that bypasses the application level auditing and access controls. In other systems, particularly data management systems, the normal mode of access is expected to be interactive. Programmers with the ability to use access method primitives can frequently access database files directly without leaving any trace in the application access control and audit logs. Under the circumstances, such audit trail concepts can do little more than attempt to detect frontal attacks on some system resource.

Security audit trails can play an important role in a security program for a computer system. As audit trails are presently structured on most machines, they are only useful primarily in detecting unauthorized access to files. For those computers which have no access control mechanisms built into the primary operating systems, the audit trail bears the burden of detecting unauthorized access to system resources. As access

control mechanisms are installed in the operating systems, the need for security audit trail data will be even greater; it will not only be able to record attempted unauthorized access, but will be virtually the only method by which user actions which are authorized but excessive can be detected.

This work offers much valuable direction in the compilation and organization of the traditional, or audit trail, class of system surveillance data. The report introduces a *surveillance system* structure. This nomenclature describes a system for the analysis of raw audit trail data.²⁸ System elements are described "for the automated generation of security exception reports" (Section 4, Structure of a Surveillance System [17]). The structure includes a selection program that operates on the raw audit trail data and certain selection parameters, plus a surveillance program that operates on certain resulting session/job records. This is used together with the audit history data to produce security exception reports.

3.5 Suspicious Event Testing and Weighted Scoring

The fifth group publishing in this new field consists of the author and his associates Robert A. Clyde and James D. Gates at Clyde Digital Systems. These papers are found in the proceedings of meetings of the Insider Threat Identification Systems Working Group [6, 7, 8 and 9]. The paper by Robert A. Clyde [8] discusses the suspicious event testing and weighted scoring used by the SentryGATE product line.²⁹ This work has been based on the analysis of system-event and user-interaction surveillance data. High scoring users are presented as an ordered list on a report provided to the security administrator. Though limited in scope, numerous acts of misconduct, including criminal conduct, have been detected on sensitive computer systems with this capability.³⁰ Tests for misconduct in the use of dangerously privileged system programs depend critically on detailed user-interaction surveillance.

For example, consider a program used to install privileged programs, or the one used to maintain access authorization tables and system passwords. With traditional system audit or accounting logs, the perpetrator need only re-name the dangerously privileged program in order to execute it undetected. Should the execution of such a program be detected, the traditional audit records will not offer any information about what the user did.³¹ For the protection of authorized users acting in good faith and within the security policy guidelines, a record

²⁸ The author has used the word *surveillance* to describe an ongoing monitoring activity of the target system. This usage is consistent with product literature that has been in the field for several years. Anderson's use describes a system that analyzes audit trail data.

²⁹ The SentryGATE product line includes the user-interaction surveillance module designed by Robert A. Clyde and implemented with assistance by his project team at Clyde Digital Systems, Orem, Utah. Some 14 suspicious event tests are included with weighted scoring and an ordered listing of suspicious users, starting with those determined to represent the highest risk of insider threat [8].

³⁰ With respect to the user-interaction surveillance now in the field, numerous reports have been received by the vendor of misconduct and criminal activity which has been detected using same on C1 and C2 rated, secure systems. In each instance, the damage to the sponsor was perpetrated by insiders. The evidence gathered by the surveillance technology includes the keystrokes used to penetrate the access controls, together with the corresponding system responses.

³¹ See for example footnotes 17 and 27 for a discussion of a recent case regarding operating system alarms and the traditional audit trails provided with operating systems now in wide use.

of keystrokes and system responses is essential. For the perpetrator of damaging activity in the use of dangerously privileged system programs, such a record is essential for the ultimate detection of the true perpetrator of the original source of compromise. For example, an authorized user, experiencing altered motivations, may give certain privileges to unauthorized persons in a collaboration of compromise and exploitation.

3.5.1 System Surveillance Selectivity

In order to effectively enforce a broad spectrum of policy requirements for system surveillance across a range of automated information systems of varying sensitivity, the surveillance technology must be high performance,³² and it must be able to meet a demand for full-trace capture of user-interactions, batch stream activities and system-events when required. To deal with these varying requirements and with what may be regarded in some cases as excessive surveillance data, the surveillance technology must support parameterization for selectivity of the monitored activities and of the data captured in response to the governing policy.

For some cases the mandatory capture of just the keystrokes of certain, or perhaps all, of the users at all times may be a suitable parameterization of the surveillance system for the policy in force.³³ At other sites, there may be a policy of blanket monitoring. There are numerous sites now using this monitoring mode with user-interaction surveillance.

3.5.2 Secure Master Control

In order to assure secure access to a master control module that is capable of parameterizing the execution of the surveillance system, and in order to assure the tamper-resistance of the surveillance system itself, two techniques are recommended: 1) the executable image of the master control program can be kept encrypted for all except a small portion of its user-interface logic, and 2) direct access to the surveillance system should not be given to the master control program, just as it is not given to the operating system, the trusted computing base or any other facility. This is in order to assure the independent tamper-resistance of the surveillance system.³⁴ It is also important that the execution of the user-interface logic require knowledge of the decryption key which is external to the system.

The parameters set by the master control module can be encrypted into a table for subsequent decryption by the surveillance system. The surveillance system itself should be brought

³² One of the most limiting problems with traditional audit trail capabilities currently delivered with the widely used operating systems as a set of discretionary functions is the inordinate burden they impose on the processor. The consequence is a persistent conflict of interest in demand for processor resource between the requirements of security policy for monitoring and the intended use of the system in meeting the sponsor's production objectives. A security policy will not be viable in its enforcement unless the technologies required for that enforcement do not conflict with the production objectives of the target system.

³³ At some sites, where monitoring activities have been practiced with the limited traditional auditing techniques, it has been suggested that considerable value could be obtained from monitoring for keystrokes only. It is suggested by some that this is sufficient to that task of successful detection of suspicious events by automated analysis of such data. However, when the system responses are not recorded, the evidence value of the raw surveillance data is reduced unrecoverably. (e.g. It is difficult to determine with certainty what took place when system responses are not captured.)

³⁴ See footnote 22 for a discussion on tamper-resistance and data hiding, together with domain independence and layering. Also, see section 4.3.1.1 on operational assurance, in reference [1].

onto the computer with media containing an executable image that is encrypted for all but a small portion of logic that provides a user-interface for the insertion of an external decryption key. With this key the surveillance system decrypts, loads and starts itself. Thereafter, it decrypts and interrogates the parameter table created by the master control module. For added protection the master control module can be removed from the system when not in use. In addition, the external decryption key necessary to run the master control module should be different from the external decryption key necessary to bring up the surveillance system.³⁵

3.6 Raw Surveillance Data

The term *raw surveillance data* is given here to unaltered surveillance data, exactly as it is captured and originally recorded by the surveillance system. Such data can be used to support a number of objectives in the management of automated information systems. Those of particular importance to a discussion of insider threat identification systems include the following:

- Detection of suspicious events by automated analysis
- Investigative activities for perpetrator identification by direct inspection
- Evidence gathering for case development
- Fact-oriented generation of a surveillance knowledge base, used for interactive, expert system identification of perpetrators
- Direct inspection for detailed assessment of known damage
- Recovery of damaged data structures
- Direct inspection and analysis of attempted penetrations for unexpected weaknesses in the access controls

It may be concluded that the generation and retention of raw surveillance data at a detailed level is of substantial value.

Indeed, other important objectives suggest themselves, and although these objectives lie outside the scope of this paper, they nevertheless have a similar dependence on detailed raw surveillance data that can be retained for subsequent inspection and use. These objectives include the following:

- Automated disaster recovery employing keystroke surveillance data gathered from user interactions
- Automated disaster recovery employing I/O-call surveillance data
- In-depth analysis of system use patterns for capacity planning
- In-depth analysis of training levels and adequacy
- Independent auditing of process results against those expected from policies and procedures (e.g., financial auditing)
- Enforcement by computer of labor-management policies governing harassment of users

³⁵ The master control concept, credited to Robert A. Clyde at Clyde Digital Systems, has been successfully implemented by members of his project 350 team.

- Employee performance and productivity measurement (as regulated by policy and employee agreement)

It seems clear that raw surveillance data should be retained, without alteration, in order to offer subsequent support to a widely ranging set of potentially important objectives in the management of automated information systems. Furthermore, on the matter of retention, the value of any particular set of surveillance data to some subsequent demand is often very difficult to predict at the time the surveillance data are captured.³⁶

3.6.1 Archiving

The archiving of raw surveillance data from blanket monitoring with user-interaction surveillance is now commonly practiced among security administrators who have this technology. They are not, for example, trying to predict which data may be of subsequent value. The number of bytes of user-interaction surveillance data generated from user to user can vary widely. However, the total byte counts for such data generated on a given system in a day's time are typically quite manageable.³⁷

No experience is yet available on the balance between detail and quantity of data in the areas of system-event and batch stream surveillance. However, it seems clear that batch stream surveillance need not be more than a modest extension to the data archiving requirements of user-interaction surveillance. It is clearly of the same character. In both cases, redundant sequences of identical data, together with large continuous outputs by the system in response to the user or the batch-command stream, can simply be counted. This count information can replace the repetitive information in the raw surveillance data set prior to archiving. Similar techniques and appropriate selection of the data captured by system-event surveillance need to be studied further.³⁸

3.6.2 Write-once Media

Optical storage technology now offers a number of competitive products in the single and multiple gigabyte range. Such products are a cost-effective solution to the archiving requirements of raw surveillance data. The write-once feature of this technology is important in assuring tamper-resistance for the surveillance record. The actual medium on which the data are written is removable from the drive mechanism. This removable medium, which is about the size of a traditional long-playing phonograph record, goes into a cartridge one inch thick. It can be packed closely and stored in quantity, even in limited vault

³⁶ Suspicious events depend primarily for their detection on the analysis of the raw surveillance data set. Detection of a suspicious event usually occurs at a time which is considerably later than the event itself. Once a suspicious event is detected, an investigation of raw surveillance data into earlier periods is always of value in establishing the chain of evidence to the original compromise.

³⁷ Data taken from a system used in a highly interactive mode indicates an average of about 250,000 bytes per user per day. This includes clerical, technical support, technical writing and development personnel. These blanket monitoring results are believed to be fairly typical of such environments. However, the amount of raw surveillance data captured in the blanket mode can vary widely, depending on the nature of user activities on a given system.

³⁸ The surveillance of I/O-call activity could become burdensome. However, in most cases it would seem adequate to capture only a small portion of read-request output without losing the ability to characterize accurately the events taking place. On the other hand, if I/O-call surveillance is to be used for automated disaster recovery, then all write activity to the data structures requiring this level of protection would have to be captured.

space. This medium is not subject to damage from electric or magnetic fields. It is expected to be stable over long periods of time, with a life of at least 10 years.

Write-once storage media require special file I/O handling that will not attempt to rewrite portions of the medium. Such a file handler must deal with the differing requirements for media faults and read access of this class of storage, in comparison with the requirements of the traditional magnetic disk technology. The file handler is supplied in some cases by the manufacturer of the optical storage drive. File handling is also provided with the surveillance system product in order to support the advantage of tamper-resistance on traditional media, where write-once can also be enforced.

3.6.2.1 Cost Effectiveness

The cost per byte of write-once media is consistent with that of conventional magnetic-tape storage media for data archiving. The cost of the drive is considerably less per byte than that of tape-oriented drives.³⁹ As for size, one optical storage drive currently offered can be fitted into 5.25 inches of standard electronic rack space.⁴⁰

In addition to the cost advantage of optical storage in archiving, this technology also offers a cost effective alternative to traditional on-line storage for random read-access to large segments of raw surveillance data. It is expected that the costs of optical storage will decline with time, maintaining a continuing advantage over magnetic media where write-once is required. For magnetic media, this write-once requirement represents an additional cost for enforcement.

The peculiar combination of cost relationships currently offered by optical storage and the expectation of declining costs are most encouraging. These benefits come at a time when it is becoming more important to archive sufficient quantities of raw surveillance data to successfully support the objectives of insider threat identification systems. Within reasonable, balanced cost constraints, it now appears that such objectives can be successfully addressed through the support of archived, full-system, raw surveillance data, where appropriate constraints are imposed on the selection of the system-events monitored and the quantity of such data captured.

3.7 Knowledge Base

The knowledge base that supports the expert system in the analysis for suspicious events is described here as a *surveillance knowledge base*. The surveillance knowledge base has a domain that includes a number of fact contributions called *fact-sets* and a number of rule contributions called *rule-sets*.

³⁹ The Perceptics Laser System optical disk storage subsystem is compared here with Digital Equipment Corporation's TK50 magnetic tape drive using a streamer tape cartridge. The Perceptics optical disk cartridge is currently priced at \$0.245 per megabyte and the Digital TK50 tape cartridge is priced currently at \$0.305 per megabyte. For the drives, Perceptics is priced at \$12.50 per megabyte and Digital is at \$35.79 per megabyte.

⁴⁰ The Perceptics optical disk cartridge measures 25mm (1 in.) in height, 330mm (13 in.) in width and 334mm (13.14 in.) in depth.

3.7.1 Fact-based Knowledge

The fact-based portion of the surveillance knowledge base includes the following distinct sets of facts:

- *Surveillance fact-set*—derived from raw surveillance data
- *External fact-set*—external to the system, including facts about changes in user status, new users and terminated users
- *Supporting fact-set*—image facts, system library facts, access authorization facts, and facts from suspicious event test modules
- *Profile fact-set*—the fact-oriented portion of the surveillance knowledge base contained in each of a number of profile structures for suspicious event testing.

3.7.1.1 Primary Data Reduction

A variety of data extraction and structuring activities may be applied to the raw surveillance data may be found useful, depending on the system management objectives. However, for support of the objectives of an insider threat identification system, only one such activity is considered here. The intention of this activity is to create an appropriately structured, fact-oriented surveillance knowledge base by suitable extraction from the raw surveillance data. The extraction process should invoke a set of rules that serve to mediate the exclusion of extraneous data in the generation of a reduced data set. This primary set of rules must be based on expert knowledge about which data are relatively safe to ignore.

3.7.1.2 Surveillance Fact-set

The reduced data are to be used in constructing the *surveillance fact-set* portion of the surveillance knowledge base. These facts are extracted from the raw surveillance data captured over a specified period of time. The construction of this contribution to the surveillance knowledge base is governed by the base-level knowledge engineering logic in the expert system. This base-level knowledge engineering is limited to that which can be independent of a given system. The fundamental element of this fact-set is called a *surveillance record*.

The surveillance record is an access-oriented structure of the following form:

(subject, object, access-attributes)

The access-attributes include the following:

- *Surveillance module source*—a surveillance module that captures the data (this is essential to each surveillance record)
- *Action*—the type of operation performed by the subject on the object
- *Resource usage*—such information as the number of reads, writes, lines printed, CPU resource used, I/O resource used, etc.
- *Date and time stamping*—the date and time of the access (this is an essential data item in each surveillance record)
- *Keystrokes*—keystrokes (for user-interaction surveillance) and pseudo-keystrokes (for batchstream

surveillance), included in the surveillance record as required

- *Subsequent action*—the response of the system to an access; that is, the action or condition that resulted from the access, and as much of the output data (information sent to a terminal or other output device) as required (the action or condition is essential data)

This structure can be characterized as an *n-tuple*, where *n* varies with the surveillance module on which the surveillance record data depend.

3.7.1.3 External Fact-set

The *external fact-set* is a collection of facts external to the system under surveillance that may contribute to the fact-oriented portion of the surveillance knowledge base. Such facts may include information about termination of employment; new users; users with changed status, including promotions, demotions and changed authorizations; and information that may characterize motivations. This information may be updated to the surveillance knowledge base from time to time as required.

3.7.1.4 Supporting Fact-set

The *supporting fact-set* contains other types of fact-oriented knowledge that are included in the surveillance knowledge base. The access authorization tables are one such type of supporting facts. These tables may take a variety of forms at varying levels of detail, depending on the system and the security policy to be enforced. These are the data used by the trusted computing base to determine if an access request is authorized. A general form of these tables is discussed elsewhere [6]. In this general form, the access of a specific object by a specific subject is supported. A copy of this class of fact-oriented knowledge (as independently secured data, extracted on a periodic basis) is a necessary complement to the surveillance knowledge base in some cases. This fact-set includes the variance from average rates of change and related measurements.

Other supporting contributions to this fact-set include changes in the operating system image and the system library image. Of interest is the detail of the change, together with the variance from average rates of change over a period of time.

In addition, there are supporting facts associated with the suspicious event test modules and their modification and extension.

3.7.1.5 Profile Fact-set

The concept of a *profile* comes from the need for a construct that can represent a norm for a specific activity on the target system. It has grown out of the statistical analysis approach to suspicious event testing. It is used here as an extended construct for supporting both statistically-oriented and inference-oriented suspicious event testing. These constructs contain activity-dependent facts that contribute to the knowledge base, both for pattern matching and for detection by inference. This contribution is the *profile fact-set*. Facts derived during the course of analysis are also included.

3.7.2 Rule-based Knowledge

The rule-oriented portion of the surveillance knowledge base depends critically on expert knowledge of system-compromising techniques and modes of security policy violation. A rule is characterized as a fundamental structural element in the surveillance knowledge base. It has the following form:

If (proposition) then (action)

When the proposition tests true, the action is performed. Establishing a new fact is an important class of action. In general, an inference-oriented suspicious event test is a set of one or more rules. Some propositions and actions are fixed, independent of the daily dynamics of a system or related external events. Other propositions and actions may vary.

Some parameters in propositions and actions may be automatically changed by an update to the fact-oriented portion of the surveillance knowledge base. Others may be changed, or new rules created, depending on the action of a previous rule. It is also important that some of these parameters be accessible to an investigative expert. The parameterization of some suspicious event tests is essential to an investigation in which the expert system is used by trained professionals in an interactive mode.

3.7.2.1 Surveillance Rule-set

The *surveillance rule-set* is associated with the surveillance fact-set. The construction of this contribution to the surveillance knowledge base is governed by the base-level knowledge engineering logic in the expert system. This rule-set is limited to just those rules that can be independent of a given system.

3.7.2.2 External Rule-set

The *external rule-set* is associated with the external fact-set. A *domain expert* is to be supported with tools for the modification and extension of the rules in this set.⁴¹ These rules are also to be parameterized where possible to support the domain expert in easily performing various investigative experiments. The parameter changes must be simple to understand in their effect and must be straightforward to perform.

3.7.2.3 Supporting Rule-set

The *supporting rule-set* is associated with the supporting fact-set. As with the external rule-set, a domain expert is to be supported with tools for the modification and extension of the rules in the set. And again, the rules must be parameterized, where possible, to permit investigation. The rules are constructed to address changes in use norms and to identify suspicious activity by inference when there have been changes in user status.

3.7.2.4 Profile Rule-set

The *profile rule-set* is associated with the profile fact-set. It includes activity-dependent rules, together with rules that may be established by the expert system and inserted during the course of analysis. The potential for such inserted rules must exist throughout the expert system.

⁴¹ Donald A. Waterman [11] characterizes a *domain expert* as: "A person who, through years of training and experience, has become extremely proficient at problem solving in the particular domain." The domain here is that of expertise in system-compromising and penetration techniques relative to specific security policy and system characteristics.

3.7.3 Profile Set

The profile set consists of a number of profile structures. The profile structure used here is an extension of that described by Denning [3] to accommodate the advantages of surveillance record data. The advantage of this type of data is the ability to look more closely at use patterns. The profile structure is described here as a *profile record*. The structure of a profile record is of this form:

(dependent components, independent components)

The following components of the profile record depend on the subject and object in a given access relationship:

- *Subject pattern*—the pattern to match with the subject string in the surveillance record
- *Object pattern*—the pattern to match with the object string in the surveillance record
- *Fact-set*—includes the results of one or more statistical tests, the parameters for the statistical model and some profile-record dependent facts as required
- *Rule-set*—includes certain profile-record dependent rules as required
- *Inference Logic*—is used in conjunction with profile-record independent knowledge as part of the expert system

The following components are independent of the subject or object in a given access relationship:

- *Variable name*—uniquely identifies the profile record for a given subject pattern and object pattern
- *Surveillance module pattern*—the pattern that matches with the surveillance module identification string in a surveillance record
- *Action pattern*—the pattern to match with the action string in a surveillance record
- *Resource usage pattern*—the pattern to match with resource usage data in a surveillance record
- *Period pattern*—the pattern matched to the duration-of-action information as derived from the date and time data in a surveillance record
- *Keystroke pattern*—the pattern to match to keystroke data in a surveillance record
- *Subsequent action pattern*—the pattern to match with subsequent action data in a surveillance record
- *Fact-set*—suspicious event test to be used and certain profile-record independent facts that result from inference or statistical analysis (this fact-set also includes the threshold parameters used in determining suspicious activity from the results of a statistical test)
- *Rule-set*—includes certain profile-record independent rules resulting from inference or statistical analysis

Each profile record represents a specific *profile* defined uniquely by the variable name, subject pattern and object pattern. The general constructs for specifying a pattern [3] include the following:

- Character string
- Wildcard matching for any string
- Match for any numeric string
- Match for any string in a given list
- The string matched with a given string is to be associated with a given name
- Match pattern 1 followed by pattern 2
- Match pattern 1 or pattern 2
- Match pattern 1 and pattern 2
- Match for all but the pattern

These constructs are used to support a variety of statistical models. A number of such models are discussed by Denning [3]. These models are used to perform statistically-oriented suspicious event testing by matching profile record patterns against selected surveillance records. The results are acted upon by rules for determining abnormality (suspicion) based on threshold and variance parameters.

The results of statistical analysis for certain profiles may be used to construct new facts and rules. These are placed in the independent fact- and rule-set components of the profile record and become part of the surveillance knowledge base on which the inference-oriented suspicious event tests operate.

3.7.3.1 Profile Record Classes

A *profile record class* is defined as one of a number of combinations of subject-group and object-group pairs. For example, there is a profile record class for actions performed by a group of one subject aggregated over all objects in a group that forms a *class*. Suspicious event tests are performed on these aggregations of profile records. Such tests may be called *class tests*.⁴²

3.7.3.2 System Profile Set

A *system profile set* should include a list of authorized programs from the certified system library, together with the following use attributes: 1) use frequency, 2) typical duration, 3) typical times of day used, 4) seasonal or periodic use, and 5) overall system burden. The typical system loading characteristics, including such information as job mix and peripheral use, may also be useful. Daily, weekly, monthly and seasonal system loading could be considered. Data movement and storage on the system could be characterized with current volumes, and with rates of change and variance. This should include movement into and out of the system.

No work in the identification of insider threat is known to have been reported for this type of profile table. It is likely that such information may nevertheless enjoy some use by a few government and military groups in identifying abnormal use patterns. System profile tables can clearly make a contribution to the fact-oriented portion of a surveillance knowledge base for expert system analysis.

⁴² A brief description is given by Denning for each generic class of profile record aggregates [3].

3.7.3.3 Program Profile Set

A *program profile set* should include such information as files accessed, processes initiated and privileged system-services requested. Typical data volumes and rates may also be included. In addition, data may be included for time and frequency of use. Attributes for I/O and compute process ratios may be useful, together with any special start-up or termination restrictions.

In some cases policy requires a bit-image checking of the entire system library against an independently secured copy of same. Such periodic testing can provide assurance against permanent loss of program certification from compromise due to an insider act, malicious or otherwise, or due to spontaneous media faults. This technique may be used to "disinfect" a system library under attack [18]. This practice will not, however, offer protection against a system penetration where a program is altered briefly for certain inappropriate objectives and then returned to its original certified condition.

3.7.3.4 User Profile Set

The *user profile set* should include a characterization of programs typically accessed and their frequency of use. In some cases, information on time of day normally used or on other time-use habits may be important. Similarly, certain complementary information about files accessed outside of fixed, certified procedures (e.g., with an editor) should be included. Also, habits in the use of various system services should be characterized.

Information about typical data volumes processed by the user under different circumstances and with different selected programs should be considered. In some cases it may be necessary to characterize use patterns at a keystroke level, particularly in the use of dangerously privileged programs and for critically sensitive files.

The inclusion of certain privileges and authorizations that may be complementary to specific object access rights found in the access authorization tables may also be required by some types of security policies. For example, a user may be authorized to change certain data only in specific ways.

Some activity profiles of particular interest may be found described by Denning [3]. Others are listed in outline form in [15]. An exhaustive list is represented by all objects managed by the automated information system considered against the possible actions with those objects. For example, read, write, change, compute, execute and move or copy.

3.8 Expert System Considerations

It is recommended that the expert system for analysis be constructed around a concept of *generic test modules*. For convenience three generically distinct test classifications are identified to span the domain of possible suspicious event tests. They are the following:

- *System Test Module*—includes suspicious event tests that consider system activity as a whole.
- *Program Test Module*—includes suspicious event tests that consider program activity.
- *User Test Module*—includes suspicious event tests that consider user activity

Each module generally consists of some arbitrary number of test submodules, depending on the security requirements of a given system. These submodules operate upon the surveillance knowledge base as components of the expert system. In general, the submodules relate to profile records and particular aggregates of profile records.

The expert system analysis for suspicious events must be able to produce a trace of the inference chain that leads to the identification of an asserted suspicious activity. Both algorithmic and heuristic rules must be supported. Suspicious events are to be weighted by the expert system and aggregated to characterize suspicious activities. These suspicious activities are given a criticality score, based on the weighted scoring of the supporting events. An ordered listing is to be produced with the highest scoring suspicious events occurring first. Supporting documentation is to be provided on demand at descending levels of increasing detail. For example, an investigator must be able to request the presentation of a trace of the inference chain for a specified suspicious activity. Other levels of detail include the surveillance file names, excerpts from the raw surveillance data, and finally, the raw surveillance data itself.

3.8.1 Code Generator Support

The submodules are to be constructed by fully automated code generation. Such generators are themselves a type of special expert system. In this case there is a special knowledge base that would correspond to each generic test module.

The user of the code generator would be a domain expert, where the expertise is in the field of insider threat identification. This expert would be supported by the code generator in specifying suspicious event tests for each generic test module. This type of capability requires a computable specification language within the framework of an expert system based code generator. This is a new field of technology that is attracting substantial attention with products that are beginning to stabilize.⁴³

The generation of test submodules can be characterized as the creation of the site variable portion of the rule-based knowledge in the expert system. This addresses the issue of cost effectiveness where the set of suspicious event tests and their individual characteristics may have to vary substantially from site to site, depending on policy and circumstances.

The code generator contributes a fact-set and a rule-set to the corresponding supporting fact- and rule-sets of the surveillance knowledge base. This includes the code generator's paradigm for each generic test module. The code generator in turn shares the surveillance knowledge base with the expert system for analysis.

Templates of all suspicious event test submodules are to be included with the code generator as the starting point for the domain expert at each site. This is the tool that supports the domain expert in creating, modifying and extending suspicious

⁴³ Computer-aided Software Engineering is now an active field in the private sector for product development. Some of these products are beginning to fulfill the historical promise of fifth generation language. The First International Workshop on Computer-aided Software Engineering, called CASE '87, was held in Boston late in May, 1987. A number of leading-edge contributors shared position papers and current technical status in a professional workshop environment. The proceedings are available. The more salient material is to be published by the IEEE. Clyde Digital Systems has advanced a product in the field which is believed to be capable of addressing the problem of supporting a domain expert in transforming the generic test shells into specialized, suspicious event tests by fully automated program generation.

event tests. This approach will contribute high efficiencies and low costs to the customization and production of site specific test submodules. The type of expert system-based code generator under discussion here makes it particularly easy for the domain expert to re-enter a specification session, make changes and regenerate correct code.⁴⁴

It appears reasonable to suggest that certain of the knowledge engineering which relates to the higher level structuring of the fact-oriented portion of the surveillance knowledge base could be programmed using the code generator.⁴⁵ Such structuring would deal with creating relationships among the elemental structures that support a particular suspicious event test submodule.

Some suspicious event tests require higher level structures in the surveillance knowledge base. The logic responsible for this task could come from the site-dependent knowledge engineering. This would be performed by the code generator in conjunction with the generation of site specific suspicious event tests.

3.9 Maintenance

The maintenance of the suspicious event submodules is provided by the code generator. It is intended that persons of less expertise than the domain expert be qualified to use the code generator for maintenance of the test submodules.

3.9.1 Change of User Status

A change of user status will often require an adjustment to a number of profile records involving the user as the subject. It is intended that this be performed by the system security administrator. The paradigms built into the code generator for suspicious event testing are to largely automate the task of making profile record changes based on a few simple designations of status change.

3.9.2 New Users

The insertion of profile records corresponding with a new user is to be performed by the system security administrator. This is to be a highly automated task using typical profile record templates.

4. Conclusions

Surveillance technology is the essential foundation of an insider threat identification system. The experience from the field for user-interaction surveillance encourages the belief that full-system surveillance can be achieved cost effectively with high performance products that do not represent an excessive burden to the system under surveillance. Acceptance of the surveillance concept has been expanding substantially in the private sector, with growing interest and installed sites throughout the Navy

⁴⁴ Substantial work has been done at Clyde Digital Systems by Stephen W. Clyde and his project team, under the direction of the author, on expert system based, fully automated code generation. The product, ProCODE, now in the field, supports changes to a specification and regeneration of code with particular ease.

⁴⁵ Current experience with fully automated code generation and a computable high level specification interface encourages the belief that at least a portion of the knowledge engineering can be done, site-specifically, by the domain expert. This is the knowledge engineering that would create code for building relationships among the structural elements of the surveillance knowledge base in support of any given suspicious event test.

and certain government agencies. The technology and price performance of the kind of on-line storage and archiving media required to support the surveillance concept are now entering the market.

Much work needs to be done to improve and extend the technologies of analysis for suspicious events. The work of Denning [3] at SRI suggests an even more successful application of statistical analysis techniques to full-system surveillance data. It seems clear that the expert system approach can be fruitful, particularly as it is extended to perform inference-oriented suspicious event testing. Code generating technologies are now stabilizing in the field can contribute substantially to the cost effectiveness of creating and maintaining site specific suspicious event test submodules.

Work needs to be done in characterizing the structure and detailed objectives of the last three components of an insider threat identification system. It is clear, however, that evidence gathering, damage assessment and recovery-support depend critically on detailed surveillance data at a keystroke level.

The government can benefit by offering the kind of support and encouragement to this new discipline that will send a clear signal to private capitol that there will be a market that justifies investment.

References

- [1] Department of Defense, Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, 1985, DoD 5200.28-STD.
- [2] Jeffrey D. Kuhn, "Research toward Intrusion Detection through the Automated Abstraction of Audit Data," *Proceedings of the 9th National Computer Security Conference*, September 1986, pp. 204-208.
- [3] Dorothy E. Denning, "An Intrusion Detection Model," *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, April 1986, pp. 118-131.
- [4] Lawrence R. Halme and John Van Horne, "Automated Analysis of Computer System Audit Trails for Security Purposes," *Proceedings of the 9th National Computer Security Conference*, September 1986, pp. 71-74.
- [5] Clyde Digital Systems, *SentryGATE—User Surveillance Software and Insider Threat Identification System*, Orem, Utah, February 1987.
- [6] Allan R. Clyde, "A Surveillance-Gate Model for Automated Information Security and Insider Threat Identification on Sensitive Computer Systems", *Proceedings of the 2nd Insider Threat Identification Systems Conference*, Rockville, Maryland, February, 1987.
- [7] Allan R. Clyde, "A Surveillance-Gate Implementation of an Extended Security Kernel", *Proceedings of the 3rd Insider Threat Identification Systems Conference*, Rockville, Maryland, April, 1987.
- [8] Robert A. Clyde, "Suspicious Event Testing and Weighted Scoring for the Analysis of a Surveillance Data Set," *Proceedings of the 3rd Insider Threat Identification Systems Conference*, Rockville, Maryland, April, 1987.
- [9] James D. Gates, "Tools for Identifying the Source of Security Breaches," *Proceedings of the 3rd Insider Threat Identification Systems Conference*, Rockville, Maryland, April, 1987.
- [10] Lawrence R. Halme, Teresa F. Lunt, John Van Horne, "Results of an Automated Analysis of a Computer System Audit Trail," *Proceedings of the Second Annual AFCEA Physical & Electronic Security Symposium and Exposition*, Philadelphia, Pennsylvania, August, 1986.
- [11] Donald A. Waterman, "A Guide to Expert Systems," Addison-Wesley Publishing Company, Inc., 1986, USA.
- [12] James P. Anderson Co., "Computer Security Technology Planning Study," *ESD-TR-73-51, vol. I, AD-758 206, ESD/AFSC*, Hanscom AFB, Bedford, Mass., October 1972.
- [13] Dorothy E. Denning and P. G. Neuman, "Requirements and Model for IDES—A Real-time Intrusion Detection System," Technical Report, Computer Science Lab, SRI International, 1986.
- [14] P. H. Wood and S. G. Kochan, "UNIX System Security," Pipeline Associates, Inc., Hayden Publishing Company Inc., Hasbrouck Heights, NJ 01985.
- [15] Allan R. Clyde, "User Surveillance System With Local Analysis," an unsolicited proposal by Clyde Digital Systems, Orem, Utah, submitted to a government agency, October 20, 1986.
- [16] Sytek, Inc., "Analysis of Computer System Audit Trails - Final Report," *Sytek Technical Report TR-86007*, Mountain View, California, May 30, 1986.
- [17] James P. Anderson Co., "Computer Security Threat Monitoring and Surveillance," Fort Washington, Pennsylvania, April 15, 1980.
- [18] Robert A. Clyde, "Defending Against Trojan Horses, Viruses and Worms," *VAX Systems Session Notes, Spring 1987 DECUS Symposium*, Nashville, Tennessee, April, 1987.

**AdaTM TECHNOLOGY/COMPUSEC INSERTION
STATUS REPORT**

Kenneth E. Rowe and Clarence O. Ferguson, Jr.

Office of Research and Development
National Computer Security Center

INTRODUCTION

In January 1975,¹ a joint-services High Order Language Working Group (HOLWG) was established by the U.S. Department of Defense (DoD) to identify requirements for DoD high order languages, evaluate existing languages against these requirements, and recommend the adoption or implementation of a minimal set of programming languages. The HOLWG developed the following series of increasingly-refined requirements documents: STRAWMAN, April 1975; WOODENMAN, August 1975; TINMAN, January 1976; IRONMAN, January 1977; and STEELMAN, June 1978. All of these documents went through a wide range of reviews from the DoD, academic, and industrial communities. A study was undertaken, with the release of TINMAN, to determine if any existing language(s) met the requirements. While it determined that no language or set of languages satisfied the requirements, the study indicated the feasibility of developing a new language to meet the requirements.

In April 1977,² an international design competition was launched, based on the IRONMAN requirements. The language design was completed in May 1979. A testing phase commenced and final revisions were made to the Ada language. Ada was accepted as a Military Standard (MIL-STD) in December 1980 and established as an ANSI standard on 17 February 1983.

The HOLWG Chairman, Lt Col William Whitaker (USAF, now retired), realized the need, during the language design phase,³ for programming support environments to be coupled with the language to ensure the improvements promised by the language. A series of three documents were evolved to address the support issues: SANDMAN, early 1978; PEBBLEMAN, mid-1978; and STONEMAN, early 1980. The STONEMAN document became the basis for Ada programming support environments (APSE's).

The Ada community was focused inward during this formative time for the Ada language. Little concern was given for the suitability of Ada for developing trusted systems. A language construct existed in the pre-1980 version of the language that could aid program verifiers in proving program

correctness.⁴ The construct was removed in the 1980 version.

Meanwhile, back at the ranch... the computer security (COMPUSEC) community was continuing to grow. In 1981, the DoD Computer Security Center was established, and COMPUSEC issues continued to get ever-increasing attention. Tremendous strides have been made in the technical areas of COMPUSEC, and this year marks the tenth anniversary for the National Computer Security Conference. Due to its formative nature, the inward-focused syndrome has affected the COMPUSEC community as well.

An inward focus is necessary to establish a core of expertise and experts; however, a concerted effort must be made to focus outward, each community to the other. While there has been a significant change in their focus over the past 2 years, a relatively dichotomous situation still exists between these two communities. We must establish a strong synergistic relationship between the Ada and COMPUSEC communities in order to effectively address the problem of using Ada for secure/trusted systems.

In the fall of 1984, the Center realized the need to address this disjuncture between the Ada and COMPUSEC communities. The Ada Technology Insertion Branch was established in January 1985 within the Secure Computer Networks Division of the Office of Research and Development at the Center. The goal of the branch was to foster expertise on the implications of using Ada for secure/trusted computer networks. To achieve this goal, the branch outlined its objectives. The first priority was to develop the necessary internal knowledge base in Ada and COMPUSEC. A philosophy of "Learning by Doing" was established as a means for developing this base. To focus the learning effort, the branch initiated the Secure Ada Protocols Project (SAPP).

SAPP

Given the previously described dichotomy, the SAPP team decided to approach the problem from a real-world perspective by implementing a secure protocol suite based on the Defense Data Network (DDN) specifications of transport control protocol (TCP), internet protocol (IP), and X.25. Further, we decided to do it in Ada. We felt that implementation of this secure protocol suite would give us a significant grasp on protocols and Ada so that

TMAda is a registered trademark of the U.S. Government, Ada Joint Program Office.

¹Grady Booch, Software Engineering with Ada (Menlo Park, CA: Benjamin/Cummings Co., Inc, 1983), pp. 14-16.

²Ibid., pp. 16-21.

³Ibid.

⁴Peter Wegner, Programming with Ada: An Introduction by Means of Graduated Examples (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1980), pp. 77-78.

we could begin addressing the problem of developing the same suite as a multilevel secure (MLS) suite. Our goal was to use Ada from beginning to end, including the use of Ada for the formal specifications and verification activity. We understood several things at that time:

- a. The experts said Ada was not useful for the development of secure/trusted software.
- b. There were problems with the language definition that must be addressed before we could complete our task.
- c. The MIL-STD documents were inaccurate, incomplete, and ambiguous with respect to the TCP and IP protocols.
- d. No one on the development team knew anything about protocols.
- e. Only two of the team members had ever written any Ada.
- f. At the beginning of the project, we did not have a validated (approved) Ada compiler.

Even with those problems facing us, we believed we could accomplish several important objectives:

- a. Train our people in Ada and protocols.
- b. Identify constructs of the language for use on secure/trusted applications.
- c. Identify constructs of the language requiring modification for use on trusted software.
- d. Develop an unambiguous, grammatical statement of the protocol standards in Ada.
- e. Push the state of the art for developing a "beyond-A1" MLS system.
- f. Develop a cadre of expertise with respect to the development of secure protocols in Ada.
- g. Introduce this technology to the outside communities for enhancement and refinement.
- h. Demonstrate that it could be done.

We defined the SAPP in three phases: Phase I would develop a working understanding of the Ada language and the selected protocol suite, Phase II would develop a demonstrable Ada language implementation of the protocol suite, and Phase III would develop a secure implementation of the protocol suite.

Phase I

After the initial staffing and planning of the branch, work proceeded in two areas of learning - the Ada language and

communications protocols. We first addressed the lack of experience in protocols. Each person was assigned the task of becoming intimately familiar with one of the major protocols (such as TCP) and acquiring a basic overall understanding of the entire protocol suite. Most of the training for protocols was on a self-study basis.

Our initial programming facilities were Telesoft and Janus/Ada subset compilers for the IBM personal computers (PC's). With the subset compilers, we also started a self-study of the Ada language. Using the Janus/Ada, we prototyped a high-level data link control protocol between two IBM PC/XT's. This phase was completed in April 1986.

Phase II

This phase began with the arrival of the RATIONAL computer system, a system solely for Ada development, in February 1986. After branch personnel received formal training in Ada as well as training on the RATIONAL, we started work on the high-level design of the protocol suite. Each component (TCP, IP, X.25) was implemented as a separate process. An Inter-Process Communication (IPC) specification, independent of operating system services, was developed to allow communications between components of the suite.

We demonstrated the completed suite (but not 100% full-featured) approximately 1 year after Phase II began. As the implementation proceeded, we encountered problems with Ada and the protocol specifications. Both TCP and IP were developed from MIL-STD's (1778 and 1777, respectively). These specifications were in a combination of state diagrams and structure declarations; the declarations were in a pseudo-Ada. Some of these were very difficult to express in true Ada. Variant record constructs needed to be used, but the protocol MIL-STD's were erroneous and contradictory. In addition, the description of variant records in the Ada Language Reference Manual (ALRM) was not clear. After overcoming these obstacles, the resulting design was very solid and provided a better (less ambiguous) specification than the original MIL-STD's protocol specifications.

The X.25 specifications used were the CCITT X.25 Recommendation and the DDN X.25 Host Interface Specification. The specifications for the data link and physical levels of X.25 are in prose. Translating these specifications into a high-level design was a painstaking activity. Even though the specifications were not complete, design decisions were easy to document, and the resulting Ada specification was very readable. The high-level abstraction capabilities of Ada were of great help in this design. For example, the ability to state user-defined data abstractions made it much easier to specify the checksum algorithm of X.25. We defined a package of polynomial math functions and specified the checksum algorithm as stated in polynomial form. Some minor changes were required for demonstration performance, but no change in the polynomial abstraction was needed.

The use of Ada in our design and implementation appears to have provided a very portable, programmatic specification of the DDN protocol suite. While it was not optimized for real-time performance, our implementation did demonstrate several of the objectives outlined earlier. We demonstrated that programmatic description is obtainable and more precise than the standards. In the area of Ada, we learned the language by using it. We used all of the constructs in the language, including generic packages with internal tasks and dynamically allocated tasks (via task types).

At the writing of this paper, we were in the process of porting our implementation from the RATIONAL to VAX/VMS with DEC Ada. Phase II of the project is coming to a close and we are proceeding into Phase III.

Phase III

This phase of the SAPP involves the prototype development of an MLS protocol suite. Work begins with the development of the Security Policy, followed by the Formal Model, Formal Top Level Specification (FTLS), and the implementation in Ada. The approach to this phase is to take a global view of the development of a trusted protocol suite. While it is important that we pursue the use of Ada throughout the entire development, we are not going to pursue basic research in the area of developing proof rules and verification systems. However, we will be working closely with those who are doing this basic research, both within and outside the Center.

There are initial issues about what is the correct policy and formal model for a network, how does the DoD Trusted Computer Security Evaluation Criteria apply, and how do you measure levels of trust outside of verification technology. We will consider the use of requirements tracing tools and software engineering principles to provide higher levels of trust. We plan to concentrate on some of these issues as work progresses.

While proceeding through this phase, Ada and Ada-based technology will be used wherever possible. We are committed to using an Ada-based specification language for the FTLS; Anna⁵ looks like an initial candidate to use. No verification environment currently exists for Ada-based languages, and our initial effort will proceed by using hand verification of our specifications. The ultimate benefits of not having to translate the FTLS to a different implementation language will offset this initial problem.

The ultimate hope is that this project will produce a complete, programmatic, MLS description of the protocol suite which has been verified down through 90% or more of the source code.

⁵ Anna is an annotated Ada language developed at Stanford University by Dr. Luckham, et al.

INTERACTIONS

As the work progresses, the branch will continue to interact with the government, academic, and industrial communities to foster interrelations with the Ada and COMPUSEC communities. We have worked with two groups that are especially worthy of note: the Kernel Ada Programming Support Environment (KAPSE) Interface Team (KIT) and the Ada Run-Time Environments Working Group (ARTEWG) of the Special Interest Group on Ada (SIGAda).

KIT

The KIT was established by a memorandum of agreement between the Army, Navy, and Air Force. As was mentioned, the DoD realized the need for programming support environments. It was also realized that a set of interfaces needed to be defined for the APSE's to the underlying operating systems that would give much greater portability to tools that were written and allow data to interoperate among the APSE's. The set of interfaces that were defined by the KIT is known as the Common APSE Interface Set (CAIS). Through our involvement with the KIT, MLS requirements were inserted into the CAIS in 1985. Currently, CAIS is the only DoD Standard that references the need for both Ada and a B3-level security.

ARTEWG

The ARTEWG is a part of the Association of Computing Machinery's (ACM's) SIGAda. Their objective is to address the issues of the runtime environment for Ada. The Ada language provides a sharp delineation between the compiler, the runtime environment, and the operating system. Many of the obstacles that need to be overcome in the verification arena are directly attributable to implementation dependencies in the runtime environment. The ARTEWG has published three papers that give tremendous insight into the Ada runtime.

"A Canonical Model and Taxonomy of Ada Runtime Environments" provides an historical perspective on the evolution of executives and operating systems to provide services to the application programs. The paper further suggests that the Ada compilation system can generate its own application specific, runtime system to run on a bare machine.

"Catalogue of Ada Runtime Implementation Dependencies" is a first pass at identifying all of the allowed options for implementing the runtime support for Ada compilers. The catalogue is going through peer reviews in the ARTEWG, the Performance Issues Working Group (PIWG), and many others. This paper is intended to be an exhaustive, authoritative list of all the runtime implementation dependencies.

"A Catalog of Interface Features and Options for the Ada Run Time Environment" (CIFO) is similar in intent to the CAIS.

Release 1.0 is a baseline document for the CIFO. This paper is geared towards providing a standard specification for a set of common

interfaces between the user and the runtime environment.

These three papers are very good in relation to the Ada runtime environments, but all three are devoid of COMPUSEC. The ARTEWG is concerned about COMPUSEC and is seeking input on the issues of security in relation to the runtime environment.

CONCLUSIONS

Concerns of the impact of Ada technology on COMPUSEC, and vice versa, are moving to the forefront with projects like the Strategic Defense Initiative (which has already decided to use at least an Ada-based Process Description Language [PDL]) and the NASA Space Station (which has decided to use Ada as the implementation language).

We challenge both the COMPUSEC and Ada communities to develop the synergistic relationship that is necessary to understand and resolve the problems of using Ada in and for trusted systems.

BIBLIOGRAPHY

Booch, Grady. "Software Engineering with Ada." Menlo Park, CA: Benjamin/Cummings Company, Inc., 1983.

"Catalogue of Ada Runtime Environment Implementation Dependencies." New York, NY: Association for Computing Machinery, Special Interest Group on Ada, Ada Runtime Environment Working Group, October 1986.

"A Catalogue of Interface Options for the Ada Runtime Environment." New York, NY: Association for Computing Machinery, Special Interest Group on Ada, Ada Runtime Environment Working Group, November 1986.

"A Canonical Model and Taxonomy of Ada Runtime Environments." New York, NY: Association for Computing Machinery, Special Interest Group on Ada, Ada Runtime Environment Working Group, November 1986.

"Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A-1983." Washington, DC: U.S. Government Printing Office.

Wegner, Peter. "Programming with Ada: An Introduction by Means of Graduated Examples." Englewood Cliffs, NJ: Prentice-Hall, Inc., 1980.

A Panel Discussion
* on
Ada and COMPUSEC

This panel presentation provides an open forum to begin an earnest dialogue on Ada and computer security (COMPUSEC) and their unique problems and concerns in relation to each other. From 1975 to 1983, two areas of concern were being stressed within the Federal Government (the Department of Defense in particular) and industry: escalating software costs and computer security. This concern culminated in the establishment of two standards. The Ada programming language became an ANSI/Military standard on 17 February 1983, and the Department of Defense Trusted Computer System Evaluation Criteria was published on 15 August 1983 as a DoD Computer Security Center guideline. In December 1986, the Criteria was accepted as a DoD standard (5200.28-STD). Both standards, as well as their ensuing policies, were developed separately from each other.

With projects like the Strategic Defense Initiative (which has already decided to use at least an Ada-based Process Description Language [PDL]) and the NASA Space Station (which has decided to use Ada as the implementation language), concerns of the impact of Ada technology on COMPUSEC, and vice versa, are moving to the forefront.

The panel members are:

Mr. Clarence Ferguson, Panel Moderator,
Chief, Ada Technology Insertion Branch, Office
of Research and Development, National Computer
Security Center (NCSC)

Ms. Virginia Castor, Director of the Ada
Joint Program Office (AJPO)

Dr. Charles McKay, Director of the NASA
Software Engineering Research Center (at the
University of Houston, Clear Lake)

Mr. Robert Morris, Chief Scientist, NCSC

* Ada is a registered trademark of the
U.S. Government, Ada Joint Program Office.

THE USE OF AdaTM IN SECURE AND RELIABLE SOFTWARE

Mark E Woodcock
Office of Research and Development
National Computer Security Center

"Am I so crazy to feel it's here prearranged?
The music must change!!"

-Pete Townshend, "The Music Must Change"
Who Are You, Eel Pie Publishing, 1977

ABSTRACT

The Department of Defense (DoD) has an obvious need for secure and reliable computing systems. Its language of choice, Ada, should be well suited to the development of these systems. Although it currently has some features which make it better suited to these tasks than most programming languages, Ada still requires a number of changes to properly fulfill its mission. The imprecise definition in Ada's Language Reference Manual renders Ada programs inconsistent from compiler to compiler and cannot be guaranteed to be reliable nor formally verified to meet the DoD computer security criteria. The Ada community must make a commitment to see that the research is completed to enable Ada to fulfill both security requirements and its own requirements.

THE PURPOSE OF ADA

In the mid-70's, the U.S. Department of Defense (DoD) noticed its increasing dependence on mission-critical software and that the cost of this software was growing rapidly. It has been estimated that as much as \$30 billion a year would be needed for software procurement.[1] Because one of the key contributors to this cost was the need to retrain programmers and rewrite programs when different languages were used, a decision was made to create a single programming language which could be used in all embedded systems (this includes all mission-critical and weapon systems).[4]

The DoD also noted that programs which were almost identical in functionality were quite often implemented completely differently because of the choice of run-time environment (compiler, operating system, etc.). What was needed was not just one programming language, but one that was consistent from implementation to implementation; an environment which would enable programs and programmers to move freely from one machine to the next.

The result was Ada. "Ada was designed with three overriding concerns: program reliability and maintenance, programming as a human activity, and efficiency" (section 1-3, [3]). It was intended to be usable in any mission-critical system, to exploit the advances being made in software engineering, and to be easily portable (within system size and speed limitations).

TM Ada is a registered trademark of the U.S. Government, Ada Joint Programs Office.

PROBLEMS

Clearly these were ambitious criteria; they remain beyond the capabilities of any existing language. While making great strides in achieving several of these goals (particularly in exploiting the newest developments in software engineering and usefulness for real-time systems), Ada tried to be too much for too many people. Although the language does not have to be radically altered, some significant changes are needed in order to completely fulfill its requirements.

1. Security

In approximately the same time period that Ada was being developed, the concepts that determine the security of a computing system were being defined. The features of the language are not directly critical to the security of the system, but they do play a key role in the reliability (whether the system operates in a manner consistent with its specifications). Since secure systems are supposed to be reliable (and the language effects the reliability), the features indirectly determine a system's security.

According to the DoD Trusted Computer System Evaluation Criteria (TCSEC) [4], the design of a system to be classified at the A1 level must be verified using one of the tools endorsed by the National Computer Security Center (NCSC) (e.g., Gypsy Verification Environment [GVE] and Formal Development Methodology [FDM]). Formal verification of a design (or a program) requires that the language used to describe it must be mathematically well-defined. (Ada, for reasons that will be described below, is not.) These tools require that a particular design language, created specifically for that system (Gypsy for GVE [5] and Ina Jo for FDM [6]), be used in order to reach the highest security classification.

The DoD now requires the use of an Ada-based Program Design Language[2], however, in designing its software systems. This criterion is not completely consistent with the TCSEC (particularly given DoD's desire not to allow waivers). Because the A1 classification does not address the issue of the implementation language, a system may be designed using one of the endorsed tools and then implemented in Ada. This is possible because no formal proof of the source code is necessary for an A1 evaluation. "Manual or other mapping of the FTLS (formal top-level specification) to the TCB (trusted computing base)

source code shall be performed to provide evidence of correct implementation." (page 48, [4]) Therefore, any implementation language may be used for an A1 system.

This will not be true of the "beyond A1" criteria when they become better defined. One of the requirements being considered for the A2 criteria is: "The TCB must be verified down to the source code level, using formal verification methods..." (page 51, [4]) Unfortunately, Ada code cannot be verified because the language is not well-defined. While we will soon reach a point where Ada will be required for use in these secure systems, Ada cannot meet the above requirement and could never be used in any phase of the development of a system intended to be evaluated at the A2 level.

2. Definition

The present version of Ada is defined by the MIL-STD 1815A.[3] The semantics of the language are "described by means of narrative rules" that are composed of technical and other terms. The technical terms' "precise definition is given in the text" and the other terms "are in the English Language and bear their natural meaning, as defined in Webster's Third New International Dictionary of the English Language." (section 1-5, [3]). This means that Ada is defined by the use of a natural language (English)—not a well-defined, mathematical language.

Natural languages have proven to be too complex to be used as the basis of a mathematical proof (that is, it cannot be done and never will be). The existence of the validation suite does not solve the problem. It cannot even guarantee that a compiler fully meets the requirements of the Language Reference Manual (LRM)[3] (not that any validation suite could), let alone be useful as a mathematical definition. Since verification is dependent on the ability to perform mathematical proofs, Ada cannot be verified as it presently exists.

Even if the effort was made to translate the definition of Ada into some precise mathematical notation (such as the denotational semantics recently completed by the Dansk Datamatik Center [DDC]), however, two significant types of problems would remain. The first is that the DoD, in its quest to leave no construct out of the language, included certain language features whose correctness may be impossible to formally prove (e.g., dynamic tasking, generics). The second problem is that allowing compilers to implement a particular feature in many different ways (sometimes even to the point that one compiler may use more than one approach depending on the situation—optimization versus normal operation) makes it impossible to know exactly what that feature really means.

Consider the following piece of Ada code:

```
package example is
function f (x:in out integer)
returns integer;
function g (x:in out integer)
returns integer;
end example;

package body example is
y:integer;
function f (x: in integer)
return integer is
begin
y:=x*x+y;
return y;
end;

function g (x: in integer)
return integer is
begin
y:=2*x+y;
return y;
end;

begin
y:=1;
end example;

y:=f(x) + g(x);
```

This is obviously a very simple example (of the order of evaluation problem), and any moderately-intelligent programmer would not generate such dangerous code. While any third-year computer science student would notice this, no Ada compiler is required to discover this problem. Since the two functions referenced in the assignment statement have side effects, the value of the assignment statement will vary dependent on the order of evaluation (left-to-right or right-to-left).

There are also much more complicated problems. When passed as a parameter, an atomic variable's (integer) value is passed (copy in). When a structure (such as an array) is passed, however, it may be passed by reference (its address is passed). Parameters that are passed by value are always protected, but parameters passed by reference may be altered by the operation of some other task (without the knowledge of the subprogram in question) which also has a pointer to the same variable.

This situation is particularly bad because the operation of the program may change from one execution to the next, not just between machines or compilers. Even protection schemes may fail, like checking the value of the variable and then performing some dangerous operation on it (this is known as the "time of check—time of use" problem). For example, a task might check if a variable is zero before using it as a divisor, but the check is no guarantee of correctness if another task may alter that location's value before it is used. This means that a program

could run perfectly when tested but still fail (or give incorrect results) during actual operation.

This scenario also raises the possibility of a significant security flaw. One user (even at the unclassified level) could write Ada code using tasking, shared variables, and call by reference which would enable him to bypass even the most secure existing computing systems and read any data (at any level) in the system. While this is not an Ada-specific problem, Ada does make it quite easy to accomplish.

Chapter 13, Representation Clauses and Implementation-Dependent Features, of the LRM [3] is a list of optional features which may be implemented by the compiler. While this list at least standardizes the variations between compiler versions, it does not change the fact that any program which uses one of these features is limited to a compiler which implements that feature.

One of the biggest problems with Ada, though, is the Run-Time Support Package. The run-time support needs of most programming languages is quite small. C's support package of a few instructions is dwarfed by the thousands of lines of code that may be required to run an Ada program. Not only does all this extra code (which is not validated) open vast new possibilities for errors, the system-specific features of each compiler's support package make portability even more difficult.

In the short run, it will be necessary to select one valid implementation for the language's features, validate the support package, and avoid using the more complex constructs to create a "verifiable" subset. (Note well that this does not necessarily require that a subset compiler be used, only that secure programs use only the appropriate subset.) Larger and larger parts of the language will be usable as verification technology becomes more robust, but the requirement for a concise mathematical definition will remain. While this will not please the hardline Ada supporters, it could enable Ada to be used for the design of secure systems.

RESEARCH EFFORTS

The NCSC, through the Consolidated Computer Security Program, is pursuing a two-pronged strategy to improve the state of Ada verification technology. The first is a short-term effort to demonstrate the feasibility of such a verification system and to give the community a starting point for discussion and future work. This effort is being contracted through the Rome Air Development Center to Odyssey Research Associates.

The second part is a long-term effort to produce a production quality system and is being contracted through the Defense Communications Agency. The first phase is a background study and technology evaluation being done by IIT Research Institute. After completion of this study, a request for proposal will be issued for the design of a complete

Ada Verification Environment (AVE), and a contract award is anticipated late this year.

A number of other efforts are underway to better define Ada. Dr. David Luckham at Stanford University continues to work on the Ada specification language, Anna, which will provide an operational definition of Ada. (Anna is also being used in several other projects.) DDC has just completed a denotational semantic definition of the language for the European Economic Community. There are also several people working on axiomatic proof rules for Ada.

WHY Ada HAS TO CHANGE

The one thing which has to be made really clear is that the changes suggested are not just for some rarely-applied or as yet non-existent security criteria. While the primary motivation for this paper is convincing the Ada community that it should prepare to meet the TCSEC security requirements, there are a lot of other good reasons for these changes to be made.

Both the Strategic Defense Initiative and the NASA Space Station project intend to use Ada as a design and implementation language. Regardless of how secure these systems turn out to be, they must be reliable. When kinetic-kill weapons and high-powered lasers start firing, you want to be very sure that they are working correctly. The only way to achieve the desired level of reliability is through formal verification.

Moreover, it would be more cost-effective over the life-cycle of the product if Ada systems are formally verified. The maintenance cost on a system that works correctly will be virtually non-existent. Should enhancements ever be needed, the design of the system will be so clearly stated and its interfaces so precisely defined that the changes could be made easily and cheaply.

It is clear that Ada has still not met its original criteria. There are programs which have been written that will compile on one validated Ada compiler but not on others. If this is possible, more subtle errors than a failure to compile are possible. This means there is no guarantee that an Ada program will be portable in a reliable way and the existing understanding of Ada is not sufficiently better than that of any other implementation language. This suggests that even on the system for which it was created, an Ada program will be no more reliable than a program written in another high-level language.

CONCLUSION: WHAT HAS TO BE DONE

The notion presently exists that Ada is truly the single, clearly-defined language that the DoD originally requested. Unfortunately, this is not true. Instead of having different version names (FORTRAN IV, 77, vanilla), we now have Ada differing by the compiler for which it was written. Even more significant is that Ada is not sufficiently well-defined to make it useable in a secure or reliable way.

Ada is not much worse than any other high-order language - it just includes all their flaws. The difference is Ada promised more, is expected to be used more widely (particularly for security), and has an organized system for change (the Language Maintenance Board).

What is needed is an acceptance by the Ada community that Ada will have to become formally defined and internally consistent if it is to remain the language of choice. Ada has made significant strides in syntax standardization and the inclusion of software engineering techniques in the language structure, but it still needs significant improvements. When the Ada community becomes convinced of this fact, the existing work to define Ada can be accelerated and goals of the future, as well as the language's original requirements, can be met.

REFERENCES

- [1] Edward Lieblein, "The Department of Defense Software Initiative—A Status Report," Communications of the ACM, V29, No. 8, Aug. 1986, p. 734.
- [2] John Schied, "The Ina Jo Specification Language Reference Manual," SDG Working Paper, 24 Jan. 1986.
- [3] Ada Joint Program Office, "Reference Manual for the Ada Programming Language," ANSI/MIL-STD-1815A (Washington, DC), 17 Feb. 1983.
- [4] Department of Defense, "Use of Ada in Weapon Systems," DoD Directive 3405.2 (Washington, DC), 2 Apr. 1986.
- [5] Department of Defense, Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83 (Washington, DC), 15 Aug. 1983.
- [6] Donald I. Good, et al, "Using the Gypsy Methodology," Institute for Computing Science, University of Texas at Austin, 6 June 1984.

An Ada Verification Environment

David Guaspari, C. Douglas Harper, Norman Ramsey

Odyssey Research Associates
1283 Trumansburg Road
Ithaca, New York 14850
(607) 277-2020

Abstract

A group at Odyssey Research Associates is building an environment for the formal verification of Ada programs. These programs will be verified against specifications written in PolyAnna, a high-order specification language based on Anna. PolyAnna and Anna use assertional reasoning, and we describe a logical foundation for the assertion language. We present a thumbnail sketch of Anna, and a list of ways we have modified Anna. We present an overview of how PolyAnna differs from other verification systems, and we review some features of Ada that must be restricted to make verification of Ada programs possible. We conclude with a prospective account of the higher-order parts of PolyAnna, and an explanation of why these are suited to Ada verification.

Introduction

One good reason to build tools for the formal verification of Ada programs is the expectation that they, and the verified programs, might be used—Ada compilers, it is assumed, will be widely available and Ada programmers as numerous as fleas. If formally verified software were in general use, one could learn whether the higher production costs of such software are justified by higher reliability and by savings in testing and maintenance. A group at Odyssey Research Associates is designing a specification language for Ada and a verification environment for proving the correctness of programs specified in that language. We plan to have a running system for a useful fragment of the specification language by the fall of 1988.

The specification language can be separated into two parts:

- A ground-floor language, based on Anna, supporting programming in the small (at the level of subprograms and packages);
- A higher-order polymorphic language (naturally enough called PolyAnna) supporting programming in the large.

The ground floor will be implementable by application and adaptation of established theory. Its expressive power is comparable to that of Gypsy or EHDM [Gypsy 86, EHDM 86]. Its

underlying logic is a logic of partial functions that correctly handles undefined expressions, and it is in this respect an advance on those systems. We have not yet studied proof checking and term rewriting in this logic, but we expect standard techniques to apply.

Full PolyAnna will require new research in higher-order languages and polymorphism. These are active and fashionable topics of research. At the end of this paper we sketch some well known reasons why a polymorphic language is especially suited for specifying Ada programs.

Our primary goal for our software is that it support incremental verification, rather than batch generation of verification conditions. We want to provide automated assistance for the programming style prominently associated with Dijkstra, Hoare, and Gries, that of developing a program and its proof hand in hand. (See, e.g., [Gries 83] or [Dijk 86].)

The ground floor

Assertional reasoning

Our strategy for verification is *assertional* reasoning about programs. An *embedded assertion* is, in effect, a comment—it stipulates that some condition (the assertion) is satisfied by the program state every time control reaches some point in the program text (the point at which it is “embedded”). The language in which these conditions are expressed is called the *assertion language*.

Assertional reasoning about programs, formally introduced in the famous papers of Floyd [Floyd 67] and Hoare [Hoare 69], is a set of techniques that reduces the proof of general properties of programs to the proof of finitely many logical formulas in the assertion language. A typical program property provable in this way is: if subprogram P is called in a state satisfying entry condition φ , then it will terminate in a state satisfying exit condition ψ .

By contrast, *transformational* methods begin with a specification of the desired behavior and attempt to rewrite it, by applying a series of meaning-preserving transformations, as an executable program. For example, one might specify a program recursively and apply an automatic transformation to rewrite the recursion as an iteration. The European Economic Community is sponsoring a consortium of several European universities in a very ambitious project, PROSPECTRA [KB 86], to build a transformational programming environment for Ada.

For our ground floor we use the assertional strategy because:

- It is well understood and therefore holds out a reasonable probability of success;
- There is an established pedagogy of writing programs assertionally, which we hope to support;
- Assertional reasoning is unlikely to be entirely avoidable—a transformational system must provide a facility for establishing new transformation rules, and the best understood way of doing that is assertional reasoning.

Writing assertions about Ada: Anna

Anna, described by its designers as “a cautious extension of Ada,” is a method of inserting formal comments into Ada programs. Most of these comments can be translated into embedded assertions. The commented Ada program is an Anna program, and it contains the full sense of the original Ada program, which is called the *underlying Ada text*. An Anna program can be transformed into an Ada program that runs the underlying text and checks many of the embedded assertions. Accordingly, Anna can be thought of as an extension of Ada with extra checking constructs, and which compiles into Ada.

Anna *object* and *type annotations* are associated with scopes. These can be interpreted as macro-instructions embedding assertions at certain points within their scopes (not merely at entry and exit points). Anna also allows *axioms*, which specify packages as abstract data types or as state machines, and *virtual text*, which can embody specification concepts or instrumentation.

Here is a typical Anna type annotation:

```
type EVEN is INTEGER;
--| where x: EVEN => x mod 2 = 0;
```

The formal comment (preceded by `--|`) says that every variable of type `EVEN` must contain an even value (whenever it contains a defined value at all). It applies to every observable state over the whole scope of the declaration of type `EVEN`. If the annotation is transformed into checking code, the code will raise the exception `ANNA_EXCEPTION` whenever a variable of type `EVEN` is assigned an odd value.

Here are some typical axioms, describing the semantics of a stack type:

```
--| axiom
--|   for all st: STACK; X: ELEM =>
--|     pop(push(st,x))=st,
--|     top(push(st,x))=x;
```

We assume that type `STACK` is a private type of a package, and that `pop`, `push`, and `top` are visible subprograms of that package. The annotation says: whenever all calls referred to terminate normally, the equations hold.

Finally, as an example of virtual text, we declare a virtual function that tells us whether an array of type `T` is sorted. The `--|` sign marks the virtual text—text which, if the comment sign were removed, would be legal Ada:

```
--: function sorted(a:T) return BOOLEAN;
--| where ...
```

After the `where` delimiter, the user enters his definition of the notion “sorted.” The user may supply a (virtual) body for this function, which could be used to test arrays for sortedness.

Anna has other annotations such as *propagation* and *context* annotations, which we don’t discuss here.

Modifying Anna

Anna is intended to support both formal verification (proof) and run-time testing. Our efforts, and our modifications of Anna, are directed exclusively toward proof.

Avoiding reduction to Ada The current Anna reference manual defines the semantics of Anna and of its assertion language by reducing them to Ada semantics [Anna 86]. Although the meaning of Anna’s assertion language need not be stated computationally, the reference manual gives the meaning of each annotation in terms of values computed by Ada code. Therefore one cannot provide the semantics for or the logic of Anna’s assertion language without first providing a semantics for Ada. We give a meaning to the assertion language that is independent of the semantics of Ada.

Unprovable annotations Certain Anna annotations cannot be proved solely from the definition of Ada, whether they’re true or not. The annotation “if the stack is full push propagates `stack_full`” is an example. Before push can execute `raise stack_full` a storage error or numeric overflow may occur. The possibility of such implementation-dependent events is, as far as the verifier is concerned, an act of God (and is equally mysterious).

We change our *interpretation* of some of the unprovable annotations to make them more useful. For example, we qualify all annotations by the implicit hypothesis that no storage error or numeric error occurs.

Erroneous behavior If one thinks of Anna as being defined by actual execution of checking code, then there is no direct way in which to state that certain executions are *erroneous* since they manifest themselves by differences in execution under different compilers. PolyAnna has safeguards built in that prevent us from certifying erroneous programs.

Partial correctness Anna’s out annotations have the interpretation: *if* the scope is exited normally, then the out annotation is true upon exit. There is no way to say that a subprogram will terminate normally. Techniques of proving termination for deterministic sequential programs are well understood, and we include them in PolyAnna.

Semantics of virtual functions In the current description of Anna the semantics of virtual functions is informal. A virtual function can be supplied with a body, if one wishes to generate checking code for annotations that refer to the virtual function. However, the body does not define the meaning of the virtual function; there is no way to say in Anna that the body “correctly” implements the annotations of the virtual function. “Consistency” is not sufficient, since a body that never terminates on any input is consistent with any annotation (excepting strong propagation annotations).

PolyAnna’s virtual functions are purely definitional entities. Properly annotated virtual functions are translated, without appeal to Ada semantics, into *constructors*, which are strings of our logical language. A well-formed constructor denotes a partial function, and associated with it are rules of inference that define the meaning of that function. For example, there is a family of constructors that define functions by recursion. The logical details are contained in [ORA 87b] and [ORA 87c].

Expressiveness We find it convenient to increase the expressiveness of the annotation language by generalizing several of the Anna constructs, including quantifiers, tests for definedness, and successor states.

Quantifiers Anna’s logical quantifiers are unusual. In Anna, “for all x , $P(x)$ ” is true if there is no value of x for which $P(x)$ is false. Otherwise, “for all x , $P(x)$ ” is false. (In Anna, quantified expressions are never undefined.) So, for example, “for all x , $x = 1/0$ ” is true. This is convenient and compact for writing equational axioms, when one means to say: the equations are true whenever all their constituent terms are defined. The rules of inference obeyed by Anna’s quantifiers are, unfortunately, not very convenient. For example, “for all x , $P(x)$ ” is not equivalent to a conjunction of the values of $P(x)$ for all possible values of x . In addition, Anna’s quantifiers are not *monotone* operators (see [Stoy 77]).

An expressive monotone language is desirable for the formulation of recursive definitions. The PolyAnna quantifiers are monotone, and are more expressive than the Anna quantifiers. We can define Anna’s quantifiers in PolyAnna’s logical language.

Definedness tests If x is a program variable, the Anna attribute x ’DEFINED has value TRUE if x has been initialized, and otherwise has value FALSE. If e is an expression that is not a variable, then e ’DEFINED is illegal.* It is convenient to make an analogy between an uninitialized variable and an expression that fails to denote a value (because its evaluation fails to terminate, terminates exceptionally, or is erroneous). In our formalism e ’DEFINED is legal for any expression.†

Successor states In Anna, the values of the local variables of a package make up the *state* of the package. Anna uses a procedural notation for naming package states: for instance, $S[P; Q]$

*Although it is straightforward to implement a test for x ’DEFINED when x is a variable, there is no general way to test an arbitrary expression for definedness short of attempting to evaluate it—but that evaluation may not terminate, may be erroneous, or may change the flow of control by raising an exception.

†We actually use a different notation.

is the name of the state that results from state S after invocation of P and Q (in that order) *provided that* both invocations terminate normally. If either terminates exceptionally, there is no Anna name for the state that results. We extend Anna so that all reachable package states have names.

Concreteness By design, Anna assertions are “close to the code.” Although a specification like “this operating system is secure” may ultimately be reducible to a large collection of embedded assertions about the relations of program variables to one another, the connection between those assertions and the original specification will be highly obscure. The difficulty of relating a comprehensible specification to what’s actually been proved about the program is well known and is common to all assertional systems. We expect to ameliorate this situation by providing some modular specification mechanisms in high-level PolyAnna, analogous to the mechanisms of LARCH [Gutttag 85].

Subordination to Ada syntax Anna’s conservative syntax, which essentially follows Ada’s, is useful in many ways: users with a knowledge of Ada encounter relatively few novelties; the possibility is left open that Ada tools may be modifiable into Anna tools; the job of generating checking code is more straightforward. On the other hand, Anna thereby inherits some of Ada’s compromises. For example, if private types are declared in a package, Ada requires that their implementations be given in the private part of the package; so, therefore, does Anna.* Such implementation details are precisely what a specification language wishes to abstract away from.

Here is a more esoteric example, for Anna aficionados. Consider the problem of constraining generic formal subprogram parameters. Let *sort* be a generic sorting package. We wish to constrain the formal parameter “<” by requiring it to represent a total order. Suppose, further, that instead of writing out the whole definition of “total order” we wish to import its definition from a predefined library of sorting concepts and theory, a generic package called *order_concepts*. To do so would require instantiating the *order_concepts* package with “<” at some point in the generic formal part of *sort*. Such an instantiation is not legal Ada (and, therefore, not legal Anna); generics are not treated as first-class objects. There is no conceptual difficulty with this instantiation; it is proscribed because it is beyond the compiler writer’s art. We expect that full PolyAnna will not be so tightly bound to Ada syntax.

Limitations

Our first implementation of the ground floor will omit concurrency, and will probably omit any special facilities for representing interaction with devices (in this we include I/O), or for generics.

Lessons learned from predecessors

We have learned a great deal from Anna and have tried to assess the ways in which it must be modified to suit our special purposes. We have also learned from extensive experience with the

*This requirement makes possible separate compilation.

FDM and Gypsy systems, and from a more modest acquaintance with EHDM. (See [FDM 80], [Gypsy 86], [EHDM 86].) We describe, below, ways in which we hope to improve on them.

Justifying assertional reasoning

To use the assertional strategy, we must provide a way to reduce specified programs to formulas in an assertion language. The method of reduction is called an *axiomatic semantics*. We must also provide sound logical rules for manipulating formulas in the assertion language. The axiomatic semantics should be justified against a mathematical definition of the meaning of the programming language, and the logical rules should be justified against a mathematical definition of the meaning of the assertion language.

At this time Ada has no mathematical definition. Although a formal definition of Ada has recently been completed, it has no official standing; it does not yet count as a semantics for Ada [DDC 87]. The definition is very complex, and we do not plan to justify our axiomatic semantics against it.

By extending and amending [Barr 84], we have formulated a many-sorted first-order assertion language in which expressions may be undefined, the collection of sorts may be declared to have any given structure of subsorts, and the domains modelling the sorts may be empty. Under a straightforward mathematical definition of the meaning of this language our rules of proof are sound and deductively complete.* The full Ada type structure (excepting task types) can be translated into this language. The translation is demonstrably correct with respect to our formal model of the Ada types. (Some restrictions apply; we do not model implementation-dependent attributes, and we forbid certain kinds of mildly pathological declarations. For example, an array declaration with index type equal to the whole of INTEGER is disallowed.)

We define the meaning of the assertion language by defining the meaning of "formula φ is satisfied in state σ on the hypothesis that the annotations Ann are true." The notion of hypothetical satisfaction is a way of modularizing proofs; the hypotheses in question are the assumptions that certain routines called on obey their specifications.

Incremental verification

The verification environments we know work in batch mode. The user writes a program, supplies appropriate assertions, and then submits it all to the system, which generates *verification conditions*: a list (sometimes large) of formulas in the assertion language whose truth is sufficient to imply that the program satisfies its specifications. The verification conditions must then be shown to hold.

The drawbacks of batch processing are well known. When a mistake is discovered the whole job must, as a rule, be resubmitted and much correct work may have to be redone. The relation between the specifications the user wrote and the verification conditions he sees may be obscure. Intermediate stages of verification condition generation are not observable and there

*Most existing systems use assertion languages with rules of inference that are logically unsound (quite independent of any considerations of programming semantics) because they handle undefined expressions incorrectly.

is no way to tell that verification conditions are getting unmanageably complex until it is too late.

Incremental compilers and syntax-directed editors are becoming more widely available and techniques for building them are becoming well-known. These techniques should allow us to build a verification environment in which incomplete programs can be partially verified, in which verification conditions are understandably related to the code, and which directly supports the methods of [Gries 83] and [Dijk 86].

Consider, for example, a simple while loop, guarded by condition b . Suppose for simplicity's sake that there is no other exit. The standard while-loop rule determines the effect of execution of the loop solely from the condition b and a user-supplied invariant I , which is restored by each circuit of the loop: the effect of the loop is characterized by the fact that on exit the invariant I remains true while the guard b is false. The user should be able to supply the guard and the invariant of a loop and then complete as much of the rest of the program as he likes, leaving the loop body to be filled in later.

The peculiarities of Ada

There is little hope of verifying arbitrary Ada programs. Surveys such as [ORA 85], [Good 80a], and [Good 80b] list many difficulties. Here we review some of the unusual features of Ada and describe what we do with them.

Program errors Ada introduces two special categories of program errors, executions that are "erroneous" or contain "incorrect order dependences." These errors needn't be caught at compile time and needn't be checked for at run time. The effect of an erroneous execution is completely undefined and that of an incorrect order dependence varies undesirably from compiler to compiler. For example, an attempt to read the value of an undefined scalar is erroneous, as is a procedure call whose effect depends on the parameter-passing mechanism. A procedure call whose effect depends on the order in which parameters are passed contains an incorrect order dependence.

We think it essential that the products of a verification environment contain neither erroneous executions nor incorrect order dependences. Forbidding these errors also simplifies the rest of Ada's semantics and should make verification more tractable. Some of these simplifications are discussed below.

Predefined exceptions We prove nothing about the predefined exceptions `STORAGE_ERROR` and `NUMERIC_ERROR`.* All verifications are qualified by the hypothesis that neither of these occurs. Since an implementation may choose not to report overflows we must assume in addition that no unreported overflow occurs.

One can in principle show that certain predefined exceptions, such as `CONSTRAINT_ERROR`, are never raised. We will always require proof that they are never raised. This forces a style on the programmer that forbids use of `CONSTRAINT_ERROR` as normal practice—e.g., as the intended exit from a loop.

*Some exceptions to this are described in [ORA 87a].

Optimizations We have shown elsewhere that, if a compiler makes full use of the allowed optimizations to reorder computations, it is impossible to guarantee such properties of the compiled code as “no variable is read before it is written” [ORA 85]. We can undertake to verify the effects only of programs executed in the standard order.

Restricting Ada

Here are some of the simplifications we wish to impose on the Ada programs we undertake to verify.

- We do not plan to include real arithmetic or the implementation-dependent parts of the language even in full-scale PolyAnna.
- As noted above, we treat the raising of all predefined exceptions as mistakes. We wash our hands of some such mistakes (`NUMERIC_ERROR`, `STORAGE_ERROR`) and require proof that the others will not occur.
- In order to prevent program errors resulting from improper parameter passing, we forbid certain instances of aliasing among the actual parameters of any procedure call (and between actual parameters and globals). When this is not immediately guaranteed by static analysis, we require a proof that the restriction has been obeyed. This has the additional advantage of greatly simplifying the logic of the procedure call proof rules.
- We adopt the Anna requirement that packages have the *hidden state* property [Anna 86], which guarantees, essentially, that the effects of package operations depend only on the parameters to the operation and on local variables of the package—not on externally visible (and therefore externally modifiable) variables or on variables global to the package. This makes it possible to specify the behavior of the package in isolation.
- In order to avoid program errors resulting from indiscriminate use of side effects we forbid the use of functions with side effects unless static analysis rules out the possibility of a program error.

Many other restrictions are matters of style as much as of programming language theory—for there is little hope of verifying completely arbitrary programs in any language. We hope that incremental verification will naturally impose on the user a style amenable to verification: a style in which one, in effect, outlines the proof of a routine and then implements the proof outline.

PolyAnna

We have so far described a system with capacities analogous to those of Gypsy or EHDm, but with certain important improvements made possible by some new theory and new techniques in building software environments. To exploit more fully the resources of Ada requires much more, and that is the purpose of full PolyAnna. Our account is necessarily prospective and is, to some extent, a firm endorsement of motherhood.

A higher-order polymorphic language

Higher type operations are implicit in Ada. A generic function, for example, can be thought of as a higher type operation that accepts types and subprograms as parameters and returns a function. A generic package is a higher-type operation that returns a package, packages being objects that belong to rather complicated types. Generics are not only higher type operations, but polymorphic: a formal parameter can be validly matched by objects of many types, or even by types themselves. The Ada attributes, included principally to facilitate the writing of generics, are also higher type, polymorphic operations.

For practical reasons Ada treats generics as macros rather than full-fledged operations. The esoteric difficulty for Anna sketched earlier, stemming from Ada’s inability to instantiate one generic at a particular place in the declaration of another, arises precisely because Ada does not treat generics as “first-class objects” on a par with variables (or even on a par with second-class objects like functions). Ada does not pursue its own logic to a natural conclusion, which would require generics themselves to be typed objects and accept as parameters: packages, package- and type-returning operations, the types of function-returning operations, et cetera. PolyAnna will, in the manner of ordinary mathematics, draw this conclusion by making all such entities first class.

Conclusion

An Ada verification environment offers the chance to put verified code into general use. We hope to have a running system, capable of verifying programs specified in a modest but useful formal specification language (a variant of Anna), by fall of 1988. It is based on well-established techniques of assertional reasoning about programs and is intended to improve on its ancestors in that the underlying logic of the assertion language is formally based and demonstrably sound, and that proofs of programs can be done incrementally, in step with their development. Exploitation of Ada’s mechanisms for abstraction requires extension of this assertion language, possibly to a language that is both higher-type and polymorphic.

References

- [Ada 83] *The Ada Programming Language Reference Manual*, US DoD, US Government Printing Office, 1983, ANSI/MILSTD 1815A.
- [Anna 86] David C. Luckham, Friedrich W. von Henke, Bernd Krieg-Brückner, Olaf Owe, *Anna: A Language for Annotating Ada Programs*, Reference Manual, 1986.
- [Barr 84] H. Barringer, J. H. Cheng, and C. B. Jones, “A Logic Covering Undefinedness in Program Proofs,” *Acta Informatica* 21, pp. 251–269, 1984.
- [DDC 87] *The Draft Formal Definition of Ada*, Dansk Datamatik Center, 1987.
- [Dijk 86] Edsger Dijkstra, *The Discipline of Programming*, Prentice Hall, Englewood Cliffs, 1986.

- [EHDM 86] J. Crow, S. Jefferson, R. Lee, M. Melliar-Smith, J. Rushby, R. Schwartz, R. Shostak, and F. von Henke, *Preliminary definition of the revised SPECIAL specification language*, SRI International, 1986
- [FDM 80] R. Locasso, J. Scheid, V. Schorr, and P. Eggert *The InaJo specification language reference manual*, System Development Corporation, TM-(L)-6021/001/00, 1980.
- [Floyd 67] R. Floyd, "Assigning meanings to programs," in *Mathematical Aspects of Computer Science*, XIX American Mathematical Society (1967), pp. 19-32
- [Good 80a] D. I. Good, W. D. Young, and A. R. Tripathi, "A preliminary evaluation of verifiability in Ada," in *Proceedings of the 1980 Annual Conference of the ACM*, pp. 218-224
- [Good 80b] D. I. Good and W. D. Young, "Generics and verification in Ada," in *Proceedings of the ACM Sigplan Symposium on the Ada Programming Language*, 1980, pp. 123-127.
- [Gries 83] David Gries. *The Science of Programming*, Springer-Verlag, 1983.
- [Gutttag 85] J. V. Guttag, J. J. Horning, and J. M. Wing, "LARCH in five easy pieces," systems Research Center, Digital Equipment Corporation.
- [Gypsy 86] D. I. Good, R. L. Akers, and L. M. Smith, *Report on Gypsy 2.05*, Computational Logic Inc., 1986.
- [Hoare 69] C. A. R. Hoare, "An axiomatic basis for computer programming," *ACM Communications*, vol. 21, no. 8, 1969, pp. 578-580, 583.
- [Hoare 85] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.
- [KB 86] B. Krieg-Brückner, H. Ganzinger, M. Broy, R. Wilhelm, U. Möncke, B. Weisgerber, A. D. McGettrick, I. G. Campbell, G. Winterstein, "Program development by specification and transformation in Ada/Anna," in *Ada: Managing the Transition*, Proceedings of the Ada-Europe International Conference in Edinburgh, ed. Peter J. L. Wallis, Cambridge University Press, 1986.
- [ORA 85] "Toward Ada verification," Odyssey Research Associates.
- [ORA 87a] Draft PolyAnna Reference Manual, version 0.1, Odyssey Research Associates.
- [ORA 87b] "More on partial logic," Odyssey Research Associates, 1987, in preparation.
- [ORA 87c] "Domains for Ada types," Odyssey Research Associates, 1987, in preparation.
- [Stoy 77] Joseph E. Stoy, *Denotational Semantics*, MIT Press, 1977.

Second, after the TCB receives the Procedure, the TCB validates it to ensure that all internal structures and pointers are self-referencing. Essentially, the procedure is viewed as untrusted by the TCB, so it is impossible for the Procedure to violate the security provisions of the model. Next, the TCB performs a discretionary access check on the databases and tables referenced in the Procedure. If any of these steps fail, the error is audited and the host is notified that the command could not complete. If these checks succeed, Query Execution continues to execute the Procedure.

When data are selected from the database, the TCB returns them through the Mandatory Security Check. Each security label on each row is compared to the login-level clearance associated with the user process. If the user's security level is greater than or equal to the security level of the row, the row is saved by Query Execution, and is returned to the host. Otherwise, the row is ignored and the selection process continues. The data row, along with the security level of the row and a row-level CRC is then returned to the host.

When data are inserted or updated in the database, the TCB uses the Update page CRC and Label code to perform the operation. The Update Page CRC and Label code computes the CRC for the updated data page and uses the user's login security level to update the row's new security label. Finally, it confirms the logical consistency of the row and the logical placement of a row on a memory page.

SUMMARY

The SYSDS design uses a reference monitor approach to system security to achieve a robust multilevel secure DBMS without sacrificing performance. It utilizes rows as the mandatory security object, and databases and tables as the discretionary security objects. This enables the system design to take advantage of existing Sybase DataServer software while introducing new security mechanisms. The newly re-architected product is a major departure from the basic DataServer architecture, but significant performance features of the commercial system have been maintained, indicating that the SYSDS approach will meet its goals of multilevel security with excellent performance.

Most of all, the SYSDS is intended to be a commercially viable system, able to be used in a number of government, military, and private sector data processing systems. The approach addresses the concept of data integrity and additionally introduces the concept of TCB integrity, since total system integrity is a major concern in any DBMS application. Because of these

points, it is felt that the SYSDS approach provides a solution to the multilevel DBMS problem.

REFERENCES

- [AFSB83] Air Force Studies Board, *Multilevel Data Management Security*, National Research Council, National Academy Press, Washington, D.C., 1983.
- [BIBA77] Biba, K. J., "Integrity Considerations for Secure Computer Systems," Technical Report ESD-RE-76-372, USAF Electronic Systems Division, Bedford, MA, April 1976.
- [BOEB85] Boebert, W. E., and R. Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies," *Proceedings of the 8th DoD/NBS Computer Security Conference*, 1985.
- [DODT83] *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD. Department of Defense Standard, December 1985.
- [HENN86] Henning, R. R. and S. A. Walker, "Computer Architectures and Database Security," *Proceedings of the 9th NBS/NCSC National Computer Security Conference*, September 1986.
- [LAND82] Landwehr, C. E., "What Security Levels Are For and Why Integrity Levels Are Unnecessary," NRL Technical Report Memo 7590-308: CL:UNI, Naval Research Laboratory, Washington, D.C., February 1982.
- [LAND84] Landwehr, C. E., C. L. Hietmeyer, and J. L. McLean, "A Security Model for Military Message Systems," NRL Report 8806, Naval Research Laboratory, Washington, D.C., 31 May 1984.
- [SCHL86] Schell, R., and D. E. Denning, "Integrity in Trusted Database Systems," *Proceedings of the 9th NBS/NCSC National Computer Security Conference*, September 1986.

COMPUTER DISASTER RECOVERY PLANNING:
A FAST-TRACK APPROACH

O. R. Pardo
Bechtel Eastern Power Corporation
15740 Shady Grove Road
Gaithersburg, Maryland 20877
(301) 258-4023

1.0 DISASTERS--LARGE AND SMALL--ARE
CONTINGENCIES

When we think of computer disasters, we tend to think of the large-scale disasters: hurricane, tornado, earthquake, or fire. Few of us are surprised to hear that most of all disabling "disasters" involve either water or fire. Several examples:

<u>Location</u>	<u>Year</u>	<u>Cause</u>
State of Pennsylvania, Harrisburg	1973	Flood (river)
Census Bureau, Maryland	1980	Sprinkler System
China Lake Weapons Center, California	1985	Flash Flood

Two of these events involved natural catastrophes, the third (Census) involved a combination of human error and mechanical failure. Outage times varied from 3 weeks (to partial restoration) to several months [8].

However, although very few computer centers have had to fully recover from a disaster as massive as those listed above, most centers have had to recover from a small disaster--again, most often resulting from human error, fire, or water [9]. In my personal experience, every data center that I have worked for or managed has had a flood, with plumbing (broken, inadequate, or nonexistent) involved in every case. (One reference reports that 90 percent of outage contingencies are caused by people--mostly by accident or ignorance.)

Because of the likelihood of an outage resulting from an event other than a catastrophe, we will refer to the "disaster" recovery planning process as contingency planning throughout the paper. This term properly emphasizes the wide range of use of this type of planning.

The need for contingency planning is widely reported [16,21]. In a key 1978 study, the University of Minnesota Management Information Systems Research Center reported that many American businesses could not stay in business if their critical computer systems were off

line for 7 to 14 days [16]. In 1985, the General Accounting Office study reported that only 9 of 25 computer systems studied had existing, tested contingency plans [15]. The requirement goes beyond business prudence into regulation in many cases. The federal government policy (Office of Management and Budget) requires a contingency plan for government facilities [15]. Recently, the Comptroller of the Currency reiterated its requirement that "national" banks have a contingency plan in place for critical information systems [19].

This paper outlines a method of implementing a contingency plan in a single, relatively short effort. The approach, called fast track, is to develop a workable plan by dealing only with the most critical systems first. This approach works best because it quickly reaches the crucial phase-- testing. It often proves less costly because the critical systems can be run on a smaller configuration than that required for all computer applications. It permits a recovery plan to be developed and tested within a year; we target for 6 months.

2.0 FAST TRACK

The fast-track approach to contingency planning is based on the principle of restricting the set of problems to be resolved wherever possible. It is predicated on three key beliefs: (1) restoration of a part of an organization's information systems will be better than none; (2) a contingency recovery plan must reach the test phase to demonstrate the full extent of an organization's vulnerability and to develop managements' confidence; and (3) a company can (in general) afford to backup only a subset of its data and applications.

2.1 Management Commitment

Management commitment is the key element to all contingency recovery planning [3]. In our recommended fast-track approach, it is the first phase. Depending upon the breadth of the computer systems being analyzed, the management to be involved will range from the head of a department (for a departmental system), to a division general manager (in a divisionalized company), to the chief executive officer

(CEO). The key characteristic of the manager to make the decision is the ability to recognize the computer application as key to the organization's (and, therefore, the manager's) health.

To gain management's commitment, the risk to the company of loss of information and/or information processing must be put in terms of the company's ability to perform or in terms of potential dollars lost. Once management recognizes this risk, they must agree to a budget for development and implementation of a contingency plan and for the ongoing maintenance of the plan (and enhancement if necessary). This is a significant challenge to many managers, who often find themselves two levels (or more) below the level of the key management to be involved and in a climate of reduced budgets.

The fast-track method isolates the critical applications and their vulnerability by applying two rules:

- Limited extent. Establish the extent of contingency at the walls of the computer room and develop the vulnerability of that room being disabled.
- Limited duration. Establish the extent of the contingency outage (to successful restoration--on site or at a restoration site) at 7 days.

By applying the first rule, the vulnerability (in terms of events per 100 years, the standard measure) is realistically high because of combination of catastrophe (major storm, flood, major fire) with the ordinary (human error, small fire, broken pipes, clogged sewer lines). By applying the second rule, the applications that are exposed are only those that are truly critical (likely to result in major financial loss to the company) and most recognizable as such by senior management [1]. A key element in identifying the applications is establishing recovery time criteria (i.e., the maximum time that the application can be out of operation).

The process of risk analysis should be conducted quickly and with as few people involved as possible. The data processing manager and his staff can develop the first draft and then confirm their findings with the managers responsible for the functions supported by the critical applications exposed. This group of managers then forms the team to confront senior management with the risk and the need for a contingency recovery plan. (Note: if this group cannot agree upon the criticality, then it is unlikely that a "sale" can be made to senior management.) Other members of the organization that may initiate the risk analysis include: the manager of security, the manager of the function at risk, or (best of all) the CEO. In all

cases, both the users of the application and data processing must be fully involved. (For details on risk analysis, see references 2 and 9.)

Once the critical applications are identified, and senior management recognizes the company's vulnerability, the risk analysis team (all managers) must gain senior management commitment to invest in a contingency plan and its ongoing maintenance. This will require a commitment to deliver a plan, ready for testing, in a fixed amount of time. We recommend 6 months, in four phases:

<u>Phase</u>	<u>Staff Required</u>	<u>Time</u>
1 - Management Commitment	2 - 5	4 weeks
2 - Workable Plan - Pass 1	2 - 5	4 weeks
Pass 2	5 - 10	5 weeks
3 - Affordable Plan	2 - 5	4 weeks
4 - Implementation & Testing	5 - 10	9 weeks 26 weeks

It is essential that this commitment be made if the plan is to be completed. Otherwise, the plan is likely to fail due to budget pressure, change in management sponsorship, or worse, disillusionment by the planning team.

2.2 Workable Plan

The next phase in the fast-track approach is to develop a workable and affordable plan. Can it be done? We believe it can, because the plan should only target to recover the applications that are truly critical (i.e., identified in the first phase) and for contingencies that are limited to the computer room itself.

["Why?" the reader exclaims. "Shouldn't we consider the secondary applications (those which must be restored within two weeks)? Shouldn't we anticipate a major catastrophe in which half the staff is unavailable to support the recovery process?" These questions are reasonable. However, the fast-track approach does not answer them at this time. Fast track is targeted to develop a workable and tested plan for the few truly critical applications in a limited "catastrophe." Once a successful plan is in place, it is the author's belief (and experience) that secondary (and even tertiary) applications can be accommodated more easily and at less expense.]

The process of developing a workable plan is described briefly below. The list of references provides several sources that provide superb detail for this phase [1,6,9,12,18,20]. The key to this plan is a two-pass approach: the first is conducted by a small team; and

the second by the team of key players identified in step 5.

resulting impacts on personnel, data transfer, telecommunications, and procedures).

1. Clearly define the range of contingencies that are being planned for. For example, if the computer room is destroyed (or made inoperable for a week or more), what is the effect on the telecommunications network, what is the effect on the onsite tape and disk storage, on the lists of work in progress, and on operating procedures?
2. Develop profiles of the critical applications: required computer peripherals, disk storage, tape drives, telecommunications requirements, unique operating system features, locally developed operating software and utilities, vendor (and third party) software. If data bases are involved: who is data base administrator; where are the lists of updates; are audit tapes used?
3. Review operations procedures, assure that they are up to date, and list changes that would minimize the exposure to any of the expected contingencies. Especially important are the offsite storage procedures [4] (e.g., storage of duplicate procedures offsite, more frequent backups of critical files, maintenance of backup records in duplicate, etc.).
4. Review the computer center, central telecommunications network components, and data library for ways in which loss of one can be segregated from the others.
5. Establish a recovery team roster made up of the key operations, systems, telecommunications, support, and applications users' supervisors and managers. Identify those that may be at risk (e.g., injured or dead) if one of the (limited) contingencies occur. For all individuals, select alternates. For those at risk, select secondary alternates. (Note: at this stage, it is all right to use the same individual for more than one role; however, care must be taken to avoid the "all eggs ..." syndrome.)
6. Develop most likely backup scenarios: hot-site, cold-site, full redundancy, mutual backup arrangement, etc. [4,18]. For each, develop a telecommunications backup concept [13]. These should be simply sketched out. At this point, the actual method of backup will not be decided. Only the fact of backup is important (and its

7. Develop an outline for the contingency recovery plan manual. During the first pass, this outline should be at least as detailed as that shown in Figure 2.01. During the second pass, draft sections are included where they can be fleshed out.

At the end of the first pass, the team is expanded to include the principal backup members identified in step 5. This team now reviews, criticizes, and modifies each product of steps 1 through 7. The result should present a workable plan containing all the elements required to get the critical applications back into operation. The result does not yet address cost, the actual method of backup, or the time it would take to recover. However, the result should identify those procedures, elements of room layout, and application requirements that will complicate the backup process.

In terms of staffing, this phase need not be too expensive. Many of the steps can be carried out in parallel. The second pass can be staffed by the additional members without removing them from their current responsibilities in most cases.

1. Introduction and Overview
2. Mobilization
 - Notification
 - Offsite storage
3. Operations Recovery
 - Organization
 - Backup facility
 - Checklists
4. Management Support Team
5. Administrative Support Team
6. Site Restoration
7. Maintenance and Testing
8. Team Directory

Figure 2.01

Contingency Recovery Manual
(sample table of contents)

2.3 Affordable Plan

The third phase of the fast-track approach develops the action plan for achieving a plan and recommends the best, affordable approach to backup. It is principally a task of cost estimating and problem simplification. It aims at

providing a backup plan that will meet the budget requirements set in the management commitment phase as well as the recovery time parameters required by the critical applications.

- Cost estimating. The possible backup plans are estimated by reviewing the available market options for backup recovery. Once these are known, the telecommunications backup costs are developed, especially those that are ongoing (e.g., dialup modems, ACCUNET reserve "local loops," additional network nodes). At the same time, offsite data storage costs are developed.
- Problem simplification. During phase 2, it is likely that several problems were identified that complicated the backup planning or added expense to the backup recovery plan. Examples of these are: telecommunications and/or data storage in the same room with the computers; special hardware for the critical application that is difficult or expensive to duplicate; locally developed software options or configuration-specific application features; and data base layout that combines time-critical data with historical or infrequently referenced data. Now, the team identifies changes that could mitigate the impact of these problems on the backup process. For example, it may be cheaper to build fire walls between the three areas of the computer room than to plan for likely destruction and total backup of all three areas; it may be easier to modify the application and to restructure the data base than to duplicate expensive hardware or restore all data within 24 hours.

In determining the full cost of the ongoing contingency plan, full cost should be considered: a Contingency Planning Manager (a minimum of one-quarter of a person); cost of updating the manuals at least annually; cost of offsite storage of critical application data; duplicate communications equipment (and data links if required); and the cost of having the backup site ready and tested (e.g., in the case of a "hot-site" this is usually a monthly fee that includes testing once or twice a year).

2.4 Implementation and Testing

At this point, we are ready to implement. The procedures and implementation of the offsite data backup, telecommunications network backup, operations changes, and applications changes can be cost justified and proceeded with on their own. Each will bring a measure of

protection that likely did not exist in full (or part) before. Only the backup site need be reviewed again for cost benefit. If the backup site fits the original forecast budget (and already committed to by senior management), approval to proceed should be forthcoming. During this process of implementation, the contingency recovery manual should be completed and readied for use.

Before proceeding on the backup implementation, the plan should be quickly reviewed by the full team for "test-worthiness." The team should agree that the plan will work as tested. As soon after the backup site is established, the plan should be tested [3,17]. During testing, an independent observer should track the test in terms of successes and failures. At the end of the test (or at the point that the test has failed), the team should be debriefed and an action plan put in place to correct the major shortcomings of the plan. A second test should then be scheduled and the plan verified.

If the budget will not cover the cost of establishing a backup method that meets the recovery time criteria or the application requirements, then the plan should be reviewed with senior management. They should agree to one of several conclusions:

- The recovery time criteria are too stringent.
- The budget is inadequate, or the applications involved are less critical than originally determined.
- The systems are too monolithic to permit backup. In this case, either redesign of the applications or redundant systems may be the only solution.

2.5 The Living Plan

At this point, the company (or organization) will have a tested and working plan. The plan should now be reviewed for its limits and most desirable extensions. Several have been discussed or alluded to previously:

- Extension to cover loss of the complete floor or building
- Extension to anticipate the loss of key personnel
- Inclusion of the remainder of the critical or near-critical applications.

These changes do not have to be made immediately, but they should be described in sufficient detail so that management can assign priorities and consider them for inclusion in future plans.

Just as important is the review of the plans for computer, telecommunications, and applications modifications. The fast-track process of contingency recovery planning should bring to light those critical factors that can be eliminated with changes to the computer and telecommunications configurations and redesign of software and data base applications. For example, future additions to the computer system can attempt to achieve redundancy in configurations, especially where the configurations can be separated by enough distance to reduce the chance of both centers being placed out of commission by a single event. In another instance, data base redesign may reduce the critical application to a more transportable size.

Finally, the incorporation of the contingency plan procedures into the day-to-day operations is essential [5,7]. As new applications are developed, they should be tested to see if they meet the requirements of being declared critical, and if so, added to the plan. In the meantime, all secondary applications should be encouraged to store copies of software and recent data offsite to ensure their recovery (if not immediate availability).

3.0 CONCLUSIONS

The need for contingency recovery planning is clear for businesses and organizations whose survival depends on computer systems. The fast-track approach provides an alternative to the traditional approach which pulls key people off their day-to-day jobs and delays the demonstration of benefit for 1 to 2 years. The fast-track approach minimizes expense and provides flexible positioning to changing computer needs. The following conclusions can be drawn from this paper.

- Disaster planning is an umbrella audit of all procedures. A tested disaster plan and its related hardware, software, and services is a form of insurance. As in the case of insurance, it does not have to be exercised to be useful. By developing the proper insurance program, a company actually increases its value. A disaster plan should be viewed as lowering risk to customers and investors.
- Developing and maintaining an operational disaster recovery plan is seen as an expensive process. However, when viewed for its insurance value, and as part of the overall data center operations process, the expense is reasonable and often has a payback: it develops confidence in your business by your customers.

- Commitment and support from top management is essential. A disaster recovery plan is recognition that the data center and the information systems supported by the center are critical corporate resources. A disaster plan, even when established in one year, is a long-term program, requiring testing and review on a regular basis.
- Critical systems identification is the first step to the detailed planning process. There are usually three or four applications that are key to a company's survival. They are the ones that must continue in operation in event of a disaster.
- The planning process is a continuous, iterative process. New and existing applications should be reviewed annually for inclusion (or removal) from the plan. The plan itself should be tested periodically and modified when necessary.

If a contingency plan is not in place today, the first plan should be put into action as soon as possible and updated iteratively until management is satisfied that adequate protection exists for critical business applications. With the trend toward increased use of automation, the dependence of critical business operations on the data center will increase. With the tendency toward distributed (departmental and work group) computing, the need for a corporate-wide understanding of contingency planning will grow.

4.0 REFERENCES

- [1] "A Practical Approach to Catastrophe Planning," Infosystems, pp. 68-69, February 1986.
- [2] J. W. Barker, "Dollars and Sense: The Economics of Contingency Planning" NEWS/34-38, August 1985.
- [3] W. Colby, "Disaster Recovery Plan? Nah...It'll never happen to us!" Infosystems, pp. 32-36, January 27, 1986.
- [4] Contingency Planning Options Protect Corporate Data Assets Computerworld (In Depth) pp. 73-74, January 27, 1986.
- [5] "Disaster Preparedness and Recovery Procedures" Datapro Reports, Datapro Research Corporation, March 1986.
- [6] Disaster Recovery In Today's Processing Environment, Critical Technology Report: No. C-9-85 Chantico Publishing Co., Inc. 1985.

- [7] E. Dugan, "Disaster Recovery Planning: Crisis doesn't equal catastrophe," Computerworld (In Depth), pp. 67-74, January 27, 1986.
- [8] "Horror Stories Part II" Information WEEK, pp. 40-45, March 16, 1987.
- [9] W. Lord Jr., The Data Center Disaster Consultant (Second Edition), Prentice-Hall, Inc., 1983.
- [10] M. Mandell, (editor) "Are you Ready For Disaster?" Computer Decisions, pp. 64-78, September 9, 1986.
- [11] H. W. Miller, "Disaster Recovery Planning," Journal of Systems Management, pp. 25-30, March 1986.
- [12] W. D. Minter, "Implementation of a Total Disaster Recovery Plan."
- [13] T. J. Murray, "Disaster Recovery Planning," Data Communications Management.
- [14] D. B. Parker, "Evaluating Data Backup Procedures and Services," Data Security Management.
- [15] "Put Disaster Plans On-Line Rather Than on the Shelf," Government Computer News, December 5, 1986.
- [16] J. R. Ritz, "An Evaluation of Data Processing Machine Room Loss and Selected Recovery Strategies," MISRC-WP-79-04, Working Paper Series, Management Information Systems Research Center, University of Minnesota, Minneapolis, June 1978.
- [17] "Testing Disaster Recovery/Contingency Plans," Datapro Reports, Datapro Research Corporation, March 1986.
- [18] S. Usdin, "Like It or Not, Plan For a Disaster Recovery," The Office, pp. 90-92, March 1987.
- [19] "U.S. Wants Contingency Planning By Banks," MIS Week, vol 8, Number 20, pp. 8, May 18, 1987 Fairchild Publications, NYC.
- [20] D. D. Walker, "Disaster Recovery Planning Inside General Electric," Journal of Information Systems Management, pp. 25-33, Fall 1985.
- [21] D. B. Wood, "Firms Build 'safety deposit boxes' for electronic data banks," Christian Science Monitor, pp. 21, Thursday, July 31, 1986.

RETURN TO NORMALCY: ISSUES IN CONTINGENCY PROCESSING

Thomas C. Judd
Assistant Director of the CPC
Federal Reserve System
Culpeper, Virginia

Howard W. Ward, Jr.
Assistant Professor
Germanna Community College
Locust Grove, Virginia

Much time and many words have been shared regarding data processing security measures and the recovery of lost data. Programs and procedures are widely publicized as to their monitoring and switching capabilities. The topic of disaster management, however, goes far beyond those measures. They are, of course, important and crucial concerns. Yet, a false sense of security may render the most elaborate plans worthless if the recovery process ignores the ability to return to normalcy. This ability must address the issues of confidence, reliability, integrity, availability, and the resumption of business functions with continuity of operations.

Certain industrial, commercial, and service organizations may rely upon a totally automated system. These entities usually include those activities which offer the consumer a limited choice of vendor. Federal agencies, major municipalities, utilities, securities, monetary services, military, and public safety agencies quickly come to mind. Smaller organizations, but still including the major industries of any given area, are much more affected by geographic locations of competitors. It is all of these groups to which this paper is dedicated.

The "cook book" approach has been used in an effort to provide a kind of checklist of things to do; and, one should not ignore the many tasks involved in contingency planning. The position here is not that such approaches are wrong nor incomplete nor inappropriate; they serve a very vital purpose. In the government, OMB Circular A-130 and the National Bureau of Standards' special publication NBS 500-134, "A Guide on Selecting ADP Back-up Processing Alternatives" are most helpful. OMB Circular A-130 establishes guidelines and authority while NBS 500-133 offers a useful contingency planning checklist. However, planning for contingencies is but the first step. The return to normalcy is, in our opinion, a more global and more critical issue. Protecting "our" data is important, but conveying to users the feeling of confidence that our contingency strategy provides continuous business functions is of a greater magnitude.

To create such a mechanism requires a commitment to the notion that contingency planning is both an attitude and a process; each requiring flexibility of thought, firmness of procedure, and commitment of resources. However difficult it may be, there must be a weighing of the cost of contingency planning and processing against the cost of not doing business. How long can a business function be suspended before the business will fail in its attempt to return to normalcy? Because contingency processing is not an inexpensive activity, it may well be that it is not for everyone. Hence, the most major decision of

all: How much is it worth to be able to restore the information processing capabilities?

Too often information processing is thought of only in terms of computer activities. Regardless of the method of processing data the adage of GIGO remains true. Providing for the gathering of data at the lowest level and the distribution of results at that same level is the test which must not be failed.

Alternate procedures for both of these processes with a realistic audit trail rests at the heart of any contingency plan. Major management decisions must be made at the highest level. The degree of involvement and commitment are the highly visible signs of the extent to which management truly wishes to secure itself against the hazards of modern times. The relatively simple issues of yesterday such as the disgruntled employee or the interruption of power sources remain as day-to-day concerns, but new and more devastating evils must be dealt with. Such evils include but are not limited to terrorism, nuclear damage, and environmental deterioration. Preoccupation with backup tapes and mainframes is being replaced by such concerns as who is left to operate that standby equipment, how are those people transported to that backup site, and by what means do our clients interact within a setting new to them; and perhaps new to us. The "hot-site" concept has introduced still a new dimension to the art of "being ready." It gains credence only in the context of a return to normalcy. Critical transaction-based functions ordinarily require immediate resumption to prevent serious business damage.

The position taken in this paper suggests that new strategies are required and that those new strategies involve a commitment of the highest steps of the management ladder. They require constant modification which implies extensive educational activities. They mandate a communication process with clients which instills confidence that we will continue to serve with no loss of integrity, that we will maintain a degree of reliability which can and will surpass that of our competitors, and that the availability of our support or backup procedures is both immediate and continuous to the extent necessary.

How are these strategies addressed? Simply put, they are addressed and effectively dealt with through the concept categorized as the Return to Normalcy. This concept does not substitute for well written programs, documentation, or procedures. Neither does it substitute for the many commercially available techniques which preserve already gathered data. The value of the Return to Normalcy concept rests in its ability to

involve, stimulate, and support. It embraces the following concepts:

1. continued senior management support for contingency as a process.
2. as close to simulated disaster as is feasible in testing contingency.
3. the ramifications of a fully implemented "hot contingency site."
4. the cost-effective use of fully implemented back-up sites.
5. issues in providing contingency backup for multiple sites.
6. the cost of quality contingency backup services.
7. returning to "business as usual."

Contingency preparedness, therefore, is not just a program; it is a process which remains effective only when it is reflective of a management attitude.

To deal with the concept of senior management support requires a bold commitment of resources and a willingness to defend that position. It is difficult to prescribe exactly what method of information processing backup is best. This is a relatively new field although the notions of stuffing the mattress, filling the safe deposit box, and burying the secret treasure map have been with us a long time. What makes today different is the need or desire to maintain continuous processing regardless of the operating environment.

How best to do this involves extensive study of the kinds of mishaps which might befall an organization and the measures which would allow uninterrupted service to customers or clients. Fundamental to these studies are the measurement of tolerable down time, delay in converting to the selected alternative, start-up time required to bring facilities up to full performance capabilities, reconciliation of data and/or transactions, and the conversion back to "business as usual".

A study of this magnitude is, in itself, time consuming and expensive. Further it requires parameters which delimit those alternatives to acceptable cost considerations. Because such a study adds nothing to the product or service and because it includes consideration of events which may be considered ludicrous by some, senior management stamina is vital.

Even after the conclusion of the study senior managements' role continues. A further decision must be made: is the recommendation acceptable or, if alternatives are presented, is one acceptable? If not, further study is the likely next step. And so the process continues. At each crossroad senior management must again test his or her degree of conviction that the cost of contingency preparedness is less than the cost of not doing business.

Among the possible alternatives, and a biased choice on the part of the authors, is the establishment of a "hot site." By definition, a "hot site" is a staffed facility capable of continuing information processing operations. However, hot is a relative term and there are varying degrees of staffing.

The many possibilities including degree of readiness and staffing make this a particularly difficult issue with which to deal. Ideally, paralleling facilities and personnel would offer maximum coverage and security. However, even under these conditions consideration is usually given only to computer equipment and related personnel. Issues of comparable importance which are often overlooked are input procedures including communication links to the client, control techniques, storage, forms, client service, and audit trail.

Perhaps the biggest problem with the fully equipped and staffed "hot site" is not the cost, as one might think, but rather the inability to keep talented employees honed to maximum efficiency. High turnover coupled with excessive training time and costs contribute to an expensive and inefficient operation.

A "hot site" with duplicate hardware but minimal staff provides a better solution. The limited staff can be challenged by performing developmental and testing functions. Training remains an important feature but it is more easily accomplished. However, a limited staff is unable to maintain full scale operations. To overcome this shortcoming, contingency teams can be employed. Such teams consist of selected key individuals, trained in specific aspects of operations, and who are regularly employed at remote sites. In the event of a catastrophe, the team members report to the "hot site" to lend support to the "hot site" staff.

An approach of this nature overcomes one of the major problems in disaster recovery: getting required personnel to the contingency site. It cannot be assumed that employees at a damaged or destroyed location can be relocated. They may be injured or may not have survived the incident. Travel may be interrupted or totally suspended. Further, when the chips are down, employees may not be willing to leave families or may be too distraught to be concerned with such things as contingency operations.

A well conceived "hot site" must also include current backup data, supplies, and an elaborate set of procedures. While each of these is important, procedures are the most critical. They are also the most difficult to maintain current. Not only must they describe what must be done, when, how, and by whom, they must also describe and prescribe under what circumstances a contingency plan should be activated and by whom?. And, further, they must blueprint how the organization repositions itself to a state called normalcy which may, in some cases, be a new or improved environment.

Despite the completeness of outfitting the contingency site, the staff training, and the detail of the procedures, no disaster recovery plan can be considered remotely secure without repeated testing. Unfortunately, most tests are limited to performing parallel operations under rather ideal conditions. Although it may appear to be courting self-imposed disaster, a part of the testing program must include a complete catastrophe-simulated situation. An unannounced date should be selected without regard to vacations, peak periods, or system conversions. This truly is a test of senior management conviction.

Carefully avoided, thus far, has been any approximation of cost and it will continue to be avoided since the number of variables is significant. Worthy of exploration is the use of the contingency facility to serve several users. Two points are obvious. One is that costs could be significantly reduced if even a few users are involved. The other is the dilemma if two or more users are struck at nearly the same time.

These are not the only considerations, however. Closely following the two major problems are issues such as training and supplies. While there may be certain similarities in computer operations, the many other tasks necessary to make an organization function present an enormous training problem. Consider, if you will, the variations in order entry applications. These points seem to lend added support to the skeletal "hot site" supplemented by contingency teams.

Certain cautions must be observed. A shared site cannot support two or more users from the same geographic area. Neither can it reasonably support multiple users with heavy demands for testing. Depending upon the supporting functions and the degree of activity between user and client, the site management tasks and responsibilities multiply dramatically.

Keep in mind that the scenario prompting these comments covers more than a mere local fire or flood. It is possible that there is no remaining staff to reinforce the contingency site. It is possible that radiation or a poisonous cloud has isolated the central site. It is possible that the target location just does not exist any longer.

What kind of person must be found to assure that operations can continue? Many adjectives might be used; intelligent, adaptable, fearless, patient, conscientious, loyal, and more. The obvious qualifications of loyalty and honesty go without saying. The security of any site is important and the "hot site" is no exception. However, there are some unique circumstances. If we assume that the "hot site" is not a commercial site but one under the control of the principal then it is likely there is little, if any, actual production taking place. If there were, of course, it is unlikely the description "hot site" would be applicable. This lack of production has a severe psychological effect on

"hot site" personnel. The pressure of daily deadlines is absent and so is a normal routine. Hence, personnel must be highly motivated so as to direct their energies toward constructive activities such as testing and training. These are key to providing a high level of operational expertise. The ability to function as a cohesive team suggests the need for a selective recruitment and dynamic development process.

Most of what has just been said has been directed toward the Contingency Center Specialist (Computer Operator). There are many other persons involved when dealing with the "hot site." Programmers face similar problems, but face the awkward situation of having to conform to objectives and specifications developed at another location. This lends some support to the notion of using the "hot site" as a developmental center for experimenting with new software or testing proposed procedures.

Probably the most difficult group of employees to adequately deal with are those performing the mundane and routine operations. Clerks, data entry and communications operators, and like positions are difficult to keep occupied except under production oriented conditions. No disrespect is intended since many, or perhaps all, may be of superior ability. But the opportunities for development and growth in such areas are relatively few. Somehow, a challenging educational program must be integrated into the "hot site" environment and it must be both interesting to the employee and beneficial to the employer. Perhaps the idea of an organizational training center utilizing the many resources of the "hot-site" is a concept worth exploring. Keep in mind we are continually referring to the ideal "hot site" which represents a duplicate of the site being backed up. Also keep in mind that the position taken here is that merely backing up the computer capabilities does not ensure that all of the usual necessary tasks and procedures which come before and after computer processing will also be backed up. The frequent assumption is that simply telling people to report to a different location using different machines will allow for continuous operations. This assumption is a luxurious approach to an unrealistic solution. Although statistics are not available, it is reasonable to assume that the probability is less than slight such total devastation would occur so as to require complete backup. But that is precisely the point; modern technology has provided the opportunity to render entire communities decimated and useless.

To this point it would appear that we are building to a crescendo which suggests that there really is no such thing as normalcy; that the potential problems are so great that no solution can exist. To remain competitive, business and industry must continue to strive for that better mousetrap which, incidentally, must cost less than those of the competitors. Can these two objectives be compatible within the context of allowing for

contingency processing? This reinforces the belief that senior management must seriously and carefully weigh the cost of not doing business.

For governmental agencies the answer may be somewhat easier. The profit motive is removed. The public good demands a kind of protection that warrants such expensive assurance. Major agencies such as the Department of Defense, Department of State, Federal Reserve, Internal Revenue Service and Social Security are illustrations of those entities which absolutely must have such security. State governments and major cities such as New York, Los Angeles, and Chicago would appear as likely candidates. But what of smaller municipalities? What ripple effects stemming from a disruption of major local governments which depend heavily on automated facilities would influence governmental functioning at higher levels? As examples, consider the impact of the loss of services provided by Dade County, Florida, and Fairfax, Virginia. With standardized recording and reporting techniques, regionalized "hot sites" with contingency teams or reserve forces perhaps the degree of chaos could be significantly reduced.

Let's now define that normalcy which has been the center of attention throughout this presentation. In most cases it would be defined as an environment which is familiar and accomplishes the original objectives in an equally efficient manner. Unfortunately, that definition may be oversimplified.

A familiar environment is essential; but familiar to whom? Certainly the employees must be able to consider the equipment and procedures familiar. This implies computers, office equipment, communication equipment, and supplies (particularly forms). But clients or customers must also find themselves in familiar territory. In many critical transaction-oriented processes, the user expects (and may require) 100 percent availability with no recognizable change in functionality. They must know how to communicate and must be made aware of any changes in the usual procedures. A portion of the problem is that too full a disclosure of the contingency process compromises its security which is part of the reason for its existence.

Of major importance is the people. If a new staff was to be created, would the original employees still be available? If a new site is created, especially at a distant location, current personnel be interested in relocating? If a contingency team has been used, would they want to return to their home site? There are no standard answers for these questions. However, one point is clear; the people problem is significant and those expected to play a role in contingency processing must be informed and reminded of the demands which may be made of them.

It is worth mentioning one more time that the crucial issue involves the lower level clerical positions; the mail distribution clerks,

the switchboard operators, file clerks, and supplies inventory clerk. These are often positions which do not have their tasks specifically defined or who have modified their job descriptions on their own to more effectively deal with their day-to-day operations. These highly important people who are frequently at the lower end of the pay scale are the Rodney Dangerfields of contingency planning. All too often they get no respect.

Returning to normalcy does not necessarily mean relocating back to the original site. One reason is that the site may no longer exist. A second reason is that the original location may no longer be inhabitable. If originally located near major clients or customers, they too may no longer exist. Hence, there may be no real incentive to return. But senior management must make this decision.

In any event, the "hot site" may not be the place in which to remain. It may be too small for full-scale operations in an on-going mode. Even if it is adequate, attention must now be given to a new back-up location. And the cycle begins to repeat itself.

Can an organization return to normalcy? At this writing an answer does not appear to be immediately forthcoming. The number of variables seem overwhelming. The extent and type of disaster; the attitudes of individuals; the availability of replacement personnel; the success of the contingency plan; and the availability of resources to permit a recovery all contribute to the dilemma.

Some aspects of contingency planning do hold up regardless of the disaster encountered. First, the most feasible kinds of operations to warrant consideration are those which deal with recordkeeping activities. Governmental entities, insurance companies, monetary and investment firms, and organizations selling services seem to be the most likely candidates.

Second, providing facilities and staff is expensive. Only those organizations with large financial resources can accommodate the financial requirements. When an organization exists in the profit-making arena, the less conservative competitor who elects to risk it may, in fact, drive out (or price out) its non-risktaking counterpart.

Governments with the ability to acquire revenue through increased taxes may be among the few who can afford this kind of protection. Even in this case, smaller governments simply may not be able to bear the pressures.

This is not to say no effort should be made, especially in the private sector, to safeguard the ability to maintain operations. What is being said is that there are limits beyond which the cost of not doing business may, in fact, be less than those of continuing in business. This truly is a point on the contingency scale which tests the entrepreneurial spirit of every organization.

A third point which is clear is that contin-

gency planning is not an activity to be taken lightly. There is probably some degree of contingency preparedness appropriate for every organization regardless of size. Clearly, the larger organization has more resources and more to lose. Poor or inadequate planning equates to no planning since it is ineffective and may, in fact, result in greater costs.

To combat this problem it is essential that the contingency planning team be composed of the most qualified individuals. The limits of their study and their recommendations must be clearly defined by senior management. Their advice and proposals should be carefully considered. Their authority to initiate action must also be well defined. Too liberal an approach may be too expensive to implement and too restrictive an approach may generate intolerable frustration.

Fourth, a plan which provides solely for the security of computer operations is unrealistic. It is window dressing designed to give the appearance of security in its shallowest form. The same may be said of a plan which addresses only the hardware (and software) of a computer system. Staffing concerns in both the computer and non-computer functions increase in complexity in a direct relationship with the magnitude of the disaster. Providing for business function personnel wherever they may be located is equally important. Further, contingency planning extends beyond a dust-covered manual. To serve its purpose it must be reviewed and regenerated at a pace at least equal to the growth of the organization it is expected to serve.

Fifth, in instances where quick conversion and near full-scale operations are required, a "hot site" is the safest choice. However, the degree of staffing has a major influence on the site's operation. Full staffing is expensive and inefficient in personnel usage. Minimal staffing cannot support contingency operations. Hence, a contingency team or reserve force is an acceptable compromise, both in providing automation capabilities as well as attending to the humanistic capabilities; i.e., the business function. Interim activities to maintain skills and morale may include application testing, documentation, and program development.

Lastly, none of the advantages of contingency processing can be realized without a continuous and active commitment on the part of senior management which may possibly surpass all prior requirements. Beyond this commitment is also an enormous degree of involvement to weigh alternatives and determine that point at which the cost of contingency exceeds the cost of not doing business. The contingency process; planning, testing and training, must be dynamic in that it remains effective only to the degree to which it matches the changing business environment.

Can an organization return to normalcy? The authors' opinion is that the extent of the contingency will determine how long a return

will take if such a return is, in fact, possible at all. Certainly without a contingency plan no return can be anticipated. The nature of conceivable disasters in today's world suggest that innovative approaches are called for. Cooperative ventures, "hot sites", contingency teams, and standardized procedures add to the viability of contingency planning. A return to normalcy (for all but the most farsighted, creative, daring and resourceful) seems to depend largely on the nature of the disaster, and the contingency threshold; where the cost of recovery meets the cost of not doing business, or providing vital public service.

ADVISORY MEMORANDUM ON OFFICE AUTOMATION SECURITY:
AN OVERVIEW

Alfred W. Arsenault

National Computer Security Center
Ft. George G. Meade, MD

Abstract

This paper presents an overview of National Telecommunications and Automated Information Systems Security Advisory Memorandum (NTISSAM) COMPUSEC/1-87, Advisory Memorandum on Office Automation Security, which was issued in January 1987. This guideline is divided into four parts. Part I is the Introduction and Statement of the Problem. Part II consists of guidance to the users of OA Systems, Part III is guidance to the ADP System Security Officer responsible for OA Systems, and Part IV provides guidance to Procurement Officers and others responsible for the procurement, disposal, and management of OA Systems and their associated magnetic media. In addition, there is an Appendix that addresses labeling of OA Systems and magnetic media. A distinction is made between OA systems with fixed media and those with only removable media. Fixed media are defined as those that are not meant to be routinely removed from the system by a user; all other media are considered to be removable. The guideline addresses responsibilities of system users, of the OA System's security officer, and of the organization that owns the system. Distinction is made between stand-alone OA systems (those physically and electrically isolated from other OA systems) and connected OA Systems (all others). Guidance is provided to the user for the secure operation of stand-alone systems, of connected systems used as terminals to mainframe computer systems, and of connected systems used as hosts on a LAN. An overview of threats, vulnerabilities, and controls is provided. While the Advisory Memorandum addresses issues in the areas of physical, personnel, emanations, communications, hardware/software, and procedural security, this paper concentrates on hardware/software security.

Introduction

On December 5, 1986, the Subcommittee on Automated Information Systems Security

(SAISS) of the National Telecommunications and Information Systems Security (NTISS) approved the publication of Advisory Memorandum on Office Automation Security as an NTISSAM (NTISS Advisory Memorandum). In January 1987, NTISSAM COMPUSEC/1-87 was signed by Lieutenant General Odom in his capacity as the National Manager for Telecommunications and Information Systems Security. The purpose of this paper is to provide an overview of that document.

History of the Document

The members of SAISS Working Group #3, which is responsible for developing computer security guidelines, believed that guidance was needed in the area of Office Automation Security. The National Computer Security Center, which was already working on an OA security guideline, was tasked with drafting a guideline that could be used by all Federal Government employees and contractors using OA Systems to process classified or sensitive, but unclassified, information. The working group provided review and input to the process at each step, from the preliminary outline to the final draft. When WG3 was satisfied with the draft guideline, it was sent to the members and observers of the SAISS and the Subcommittee on Telecommunications Security (STS) for their review. After several iterations of this process, the guideline was approved.

Structure of the Document

Advisory Memorandum on Office Automation Security, henceforth referred to as "the Guideline", is divided into four parts. Part I is the Introduction and Statement of the Problem. Part II provides guidance to the users of OA Systems. Part III is guidance to the ADP System Security Officer responsible for OA Systems, and Part IV provides guidance to Procurement Officers and others responsible for the procurement, management, and/or disposal of OA Systems and their associated magnetic media.

These four parts are subdivided into ten chapters that, when taken together, address all of the major security issues associated with OA Systems.

Additionally, the document provides an Appendix that addresses labeling of OA Systems and magnetic media, and a glossary of terms used in the Guideline.

The purpose of this paper is to discuss each section of the Guideline, and to describe in certain cases why recommendations were or were not made.

Introduction and Overview

To start with, we must decide what is and what is not an "Office Automation System". The Guideline defines an OA System as "Any microprocessor-based AIS or AIS component that is commonly used in an office environment. This includes, but is not limited to, Personal Computers, Word Processors, printers, and file servers. It does not include electric typewriters, photocopiers, and facsimile machines".[1] While this author readily admits to sometimes finding it hard to make a generic distinction between an "electronic typewriter" and a "word processor", it is thought that this definition makes the distinction clear in most cases.

The next step is to define the "OA Security problem". That is, what exactly are we attempting to protect?

The answer to this question has several parts. First of all, we are trying to protect information from unauthorized disclosure. United States Government policy requires that certain types of information not be disclosed to anyone unless that person has an appropriate security clearance, and/or specifically needs the information to do his or her job.[2,3] Most current OA Systems do not provide the hardware/software security necessary to enforce the separation of users and information within the system; therefore, procedure, personnel, and physical security measures must be taken to prevent unauthorized personnel from accessing the system, or from gaining access to magnetic storage media used in the system.

Secondly, we are trying to prevent unauthorized modification of information. To do this, we must again control access to both

the system and its storage media.

Thirdly, we are attempting to prevent (intentional or careless) damage to the OA system itself. This requires following a few simple rules that will help prevent the system from being either stolen or damaged.

Fixed Media vs. Removable Media

In order to make the problem easier to deal with, we make a distinction between OA systems with fixed media and those with only removable media. Fixed media are defined as those that are not meant to be routinely removed from the system by a user. Examples of fixed media are fixed disks and nonvolatile memory expansion boards. Examples of removable media include floppy disks, cassette tapes, or removable hard disk cartridges.

The type of media employed within an OA system affects what can be done with that system. Systems with only removable media can be used to process information of different sensitivity levels at different times ("periods processed," if you will). This means that information can be processed on the system that not all users of the system have a clearance, authorization, or need-to-know for. All that is required is that the information be removed from the system before these people use it.

Systems with fixed media can normally only be used to process one level of information, because the information cannot be removed from the system. Therefore, all system users must have a clearance and authorization for all information on the system.

User Responsibilities

All users must realize that they play a vital role in maintaining the security of an OA system. In fact, the role played by users of OA systems is much greater than that played by users of mainframe systems, because there are usually not as many "security professionals" overseeing what is done. Users should normally be responsible for the following, as a minimum:

(a) Knowing who the ADPSSO is for each system, and knowing how to contact that person;

(b) Being aware of, and following, all applicable security guidelines.

(c) Reporting to a security officer and known or suspected security violation. Violations of particular importance are those involving compromise or modification of information, and theft of property.

(d) Using only approved software. Software should not be used without having been tested by some responsible party (such as a security officer). Under no conditions should pirated software be used.

Operational Security for Stand-Alone Systems

A stand-alone OA system is one that is physically and electrically isolated from other OA systems. Some rules to follow when using a stand-alone system with only removable media are:

(a) Place monitor screens, printers, or other devices that produce human-readable output where they cannot be seen by casual passersby.

(b) Do not leave an OA system running unattended while it contains information that someone with physical access to it should not see. Especially, do not leave a system unattended while sensitive information is displayed on the screen.

(c) Do not leave printers unattended while sensitive information is being printed, unless the area in which the printer is located provides adequate physical security.

(d) Remove output from printers at the earliest possible time.

(e) Ensure that all human-readable outputs are appropriately marked for sensitivity. If necessary, the user should apply the labels himself.

(f) Do not eat, drink, or smoke while using an OA system.

(g) Protect magnetic media from exposure to smoke, dust, magnetic fields, and liquids.

(h) When a user is through with the system, (s)he should remove all sensitive information from it. It is also advisable to power the system off. This way, there is little or no possibility that the next user can gain access to information, no matter who (s)he is.

(i) At the end of a shift (or workday), remove all media from the system, then overwrite the system's memory with some pattern before the system is powered off. If there is a key, remove it and store it in a secure place until the next shift or working day. Remove printer ribbons that have been used to print sensitive information, and store or dispose of them.

For systems with fixed media, the above rules also apply. The main thing to keep in mind is that the sensitivity level of the system as a whole cannot normally be lowered. Therefore, users should never be allowed access to the system without clearance, authorization, and need-to-know for all information on the system.

Sensitive information can and should be removed from the system, however. When a user is finished, and has some files that contain information that should not be seen by other system users, these files should be copied to a volume of removable media, then erased from the fixed media. (Note: for most systems, use of the "delete" command will remove the information from the medium. The locations in memory must be explicitly overwritten.)

Operational Security for Connected Office Automation Systems

A connected OA System is one that is not a stand-alone. Normally, these systems are used in one (or both) of two configurations: as a terminal attached to a mainframe, or as a host on a local area network (LAN).

When an OA System is used as a terminal, it can create security problems for the system it is attached to. One of the more lucrative attacks is for a penetrator to program an OA system to copy any password that a user types. Then, the penetrator returns later and can log into one (or more) mainframe computers as one of his (or her) innocent victims. The best solution to this attack is to use only communications software that has been tested and approved by a "trusted party" (such as a security officer), and to prevent unauthorized personnel from accessing the OA system at all.

When an OA System is used as a host on a LAN, its inability to provide adequate hardware/software protection becomes more important. In most of today's OA Systems, any information contained in the system can

be accessed by anyone who can access the system. This includes anyone who can access the OA System via a network. Now, users must be extra careful not to leave sensitive information on the OA System.

Responsibilities of the ADPSSO

There should be one individual responsible for the security of each OA System. This individual may or may not be one of the users of the system. There may be a different ADPSSO for each OA System, or one with jurisdiction over all. Regardless, the ADPSSO should have the following responsibilities, as a minimum:

- (a) Ensuring that each OA System is certified and accredited, if required by organization policy.
- (b) Ensuring that all users of the system are aware of the security requirements, and assuring that all procedures are followed.
- (c) Investigating all reported or suspected security violations.
- (d) Reporting violations to appropriate authorities (e.g., top management, law enforcement officials, etc.).
- (e) Ensuring that the configuration management program is followed.
- (f) Reviewing the audit logs (if audit logs are used).

Threats, Vulnerabilities, and Controls

A threat is a person, thing, or event that can exploit a vulnerability of the system, such as a wiretapper, a business competitor, or a maintenance person.

A vulnerability is an area in which an attack, if made, is likely to be successful. Examples of vulnerabilities include lack of identification and authentication schemes, lack of physical access controls, and lack of communications security controls.

A security control is a step that is taken in an attempt to reduce the probability of exploitation of a vulnerability. Examples of controls include the use of encryption, a configuration management program, or a hardware/software security feature.

There are many threats and vulnerabilities

associated with OA Systems. These occur in the areas of physical and personnel security, communications security, emanations security, hardware/software security, and magnetic remanence. While the Guideline addresses all of these issues to some degree, we now concentrate on hardware/software security.

OA Systems can be broken down into three categories: single user systems, shared-use systems, and multi-user systems. Single user systems are those that are used exclusively by one person. Obviously, no hardware/software security is needed for these systems, regardless of whether or not fixed media is employed.

Shared-use systems are those that are used by more than one person; however, only one uses the system at a time. Multi-user systems are those that are used by more than one person at the same time. For shared-use systems, no hardware/software security is needed if only removable media is used. However, if fixed media is employed, then either all users of the system must have a clearance and need-to-know for all information, or the system should meet the requirements of at least class C1, as specified in the TCSEC. Multi-user systems should meet these requirements, regardless of whether or not they employ fixed media.

Currently, there are a large number of products available on the commercial market that claim to provide security for OA Systems. However, as of the time of this writing, none of these products has been certified by the National Computer Security Center as meeting even the class C1 requirements. While many of these security products are useful and do provide some protection, anyone using them should be careful not to be lulled into a "false sense of security."

There are several equipment vendors who are attempting to build OA Systems or workstations that will meet specific levels of the TCSEC. If these vendors are successful, it will be possible to control sharing of information on the OA System itself, by using the hardware/software controls provided by the OA System. The procedural controls needed will then be less severe than what is currently required.

Organizational Responsibilities

The organization which "owns" (or leases, or

is otherwise responsible for the secure operation of) an OA System has several responsibilities. These include:

(a) Having a security policy that defines, at a minimum, what actions are permissible on an OA System, what information may be processed when and by whom, what the organization permits regarding the use of government-owned OA Systems offsite, the use of personally owned OA Systems to do government work, procedures for maintenance of OA Systems, and procedures for the secure handling, marking, storage, and disposal of sensitive information.

(b) Setting up a training program to ensure that users and ADPSSOs are aware of their responsibilities.

(c) Having a policy concerning the procurement and use of hardware/ software. This policy should explicitly address the topics of copyrights and licensing agreements.

(d) Having a configuration management program in place.

(e) Having a policy concerning the use of audit trails.

(f) Having a policy covering certification and accreditation of OA Systems.

Procuring OA Systems

Before an organization begins to procure OA Systems, it should take several steps to determine exactly what the security needs will be. The first of these steps is a risk analysis, as defined in OMB Circular A-130.[5] In addition, the following issues should be addressed:

(a) If the OA System will be processing classified information, there are policy requirements for communications security and emanations security that must be met.

(b) Since an OA System is generally considered to be a high-dollar asset, it should be either kept in an area where it will not be stolen, or it should be locked to a table or in a cabinet.

(c) Any nonvolatile parts of the OA System should be identified.

(d) Security requirements of any Automated

Information Systems that will be connected to the OA System should be considered.

Conclusion

This guideline provides an important first step in assuring Office Automation security for the Federal Government and its contractors. It is very useful by the private sector, also.

REFERENCES

1. National Telecommunications and Information Systems Security Advisory Memorandum (NTISSAM) COMPUSEC/1-87, A Guideline on Office Automation Security, January 1987.
2. Executive Order 12356, National Security Information, 6 April 1982.
3. Public Law 93-579, "Privacy Act of 1974," 31 December 1974.
4. DoD 5200.28-STD, Department of Defense Trusted Computer System Evaluation Criteria, December 1985.
5. Office of Management and Budget (OMB) Circular A-130, "Management of Federal Information Resources", 12 December 1985.