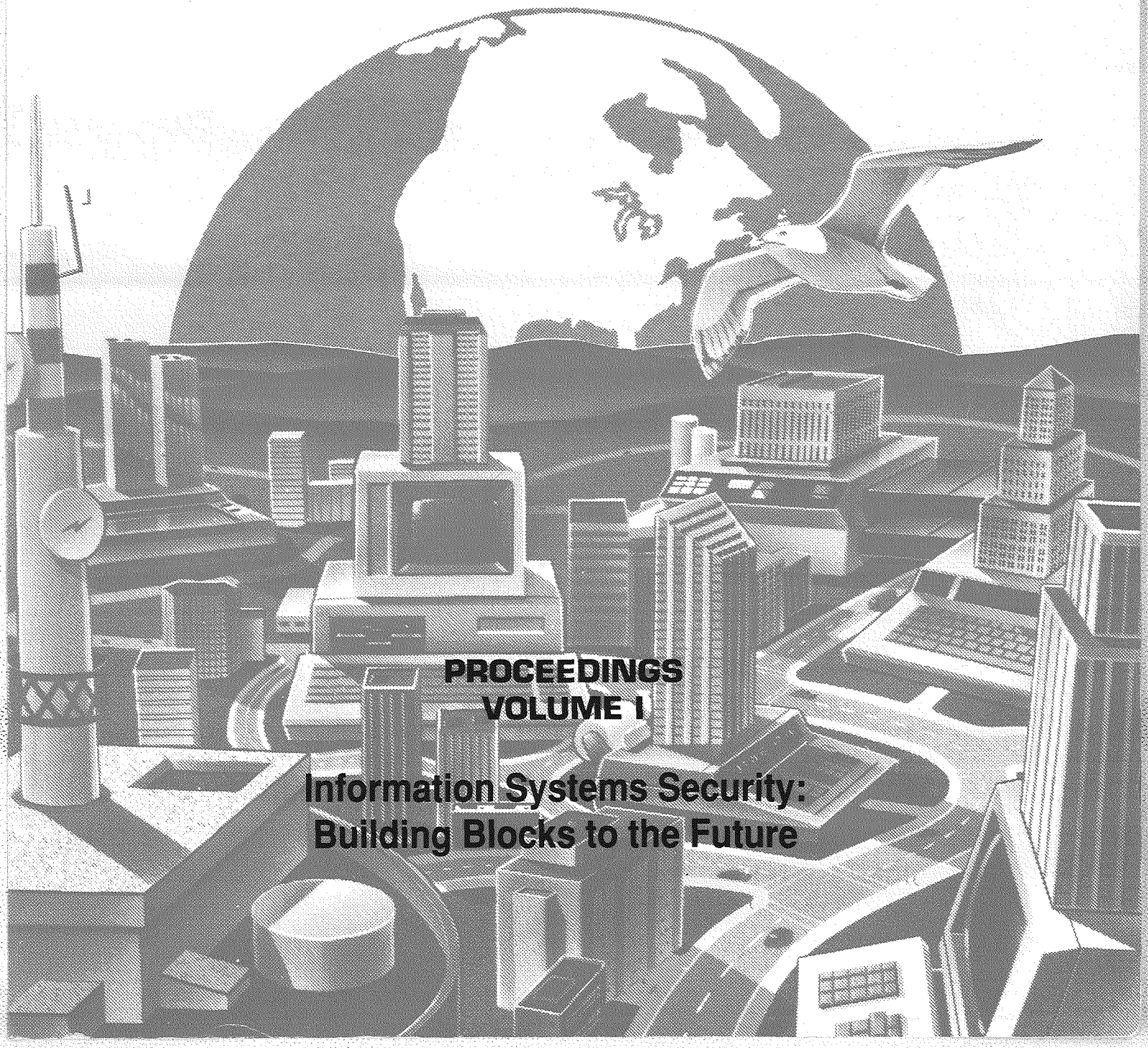
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY/
NATIONAL COMPUTER SECURITY CENTER

15TH NATIONAL COMPUTER SECURITY CONFERENCE

October 13-16, 1992
Baltimore Convention Center
Baltimore, MD



PROCEEDINGS
VOLUME I

Information Systems Security: Building Blocks to the Future

Welcome!

The National Computer Security Center (NCSC) and the Computer Systems Laboratory (CSL) are pleased to welcome you to the Fifteenth Annual National Computer Security Conference. We believe that the Conference will stimulate a vital and dynamic exchange of information and foster an understanding of emerging technologies.

The theme for this year's conference, ***"Information Systems Security: Building Blocks to the Future,"*** *reflects the continuing importance of the broader information systems security issues facing us. At the heart of these issues are two items which will receive special emphasis this week--Information Systems Security Criteria (and how it affects us), and the actions associated with organizational accreditation. These areas will be highlighted by emphasizing how organizations are integrating information security solutions. You will observe how Government, Industry, and Academe are cooperating to extend the state-of-the-art technology to information systems security. Presentations will provide you with some thoughtful insights as well as innovative ideas in developing your own solutions. Additionally, panel members will address how they develop their automated information security responsibilities. This cooperative educational program will refresh us with the perspectives of the past, and will project directions of the future.*

We firmly believe that awareness and responsibility are the foundations of all information security programs. For our collective success, we ask that you reflect on the ideas and information presented this week; then share this information with your peers, your management, your administration, and your customers. By sharing this information, we will develop a stronger knowledge base for tomorrow's journey.

PATRICK R. GALLAGHER, JR
Director
National Computer Security Center

JAMES H. BURROWS
Director
Computer Systems Laboratory

Conference

Dr. Marshall Abrams	*The MITRE Corporation*
Roland Albert	*Department of Defense*
James P. Anderson	*J.P.Anderson Company*
Devolyn Arnold	*Department of Defense*
James Arnold	*Department of Defense*
V.A. Ashby	*The MITRE Corporation*
David Balenson	*Trusted Information Systems, Inc.*
Dr. D. Elliott Bell	*BBND*
James W. Birch	*Secure Systems, Inc.*
W.Earl Boebert	*Secure Computing Technology Corporation*
Edward Borodkin	*National Computer Security Center*
Dr. Martha Branstad	*Trusted Information Systems, Inc.*
Dr.Blaine Burnham	*Department of Defense*
Dr. John Campbell	*Department of Defense*
David Chizmadia	*Department of Defense*
Dr. Deborah Cooper	*Unisys*
Donna Dodson	*National Institute of Standards and Technology*
Dr. Deborah Downs	*The AEROSPACE Corporation*
David Ferraiolo	*National Institute of Standards and Technology*
Ellen Flahavin	*National Institute of Standards and Technology*
L. Dain Gary	*Carnegie Mellon University*
William Geer	*AFCSC*
Virgil Gibson	*Grumann Data Systems*
Dennis Gilbert	*National Institute of Standards and Technology*
Irene Gilbert	*National Institute of Standards and Technology*
Captain James Goldston, USAF	*AFCSC*
Dr. Joshua Guttman	*The MITRE Corporation*
Dr. Grace Hammonds	*AGCS, Inc.*
Douglas Hardie	*Unisys Corporation*
Ronda Henning	*Harris Corporation*
Dr. Harold Highland, FICS	*Compulit, Inc.*
Jack Holleran	*National Computer Security Center*
Hilary H. Hosmer	*Data Security, Inc.*
Russell Housley	*XEROX Information Systems*
Howard Israel	*AT&T Bell Laboratories*

Referees

Professor Sushil Jajodia	*George Mason University*
John Keenan	*CISEC*
Dr. Richard Kemmerer	*University Of California, Santa Barbara*
Dr. Steven Kent	*BBN*
Richard Kuhn	*National Institute of Standards and Technology*
Steven LaFountain	*Department of Defense*
Paul A. Lambert	*Motorola GEG*
Dr. Carl Landwehr	*Naval Research Laboratory*
Robert Lau	*Department of Defense*
Dr. Theodore M.P. Lee	*Trusted Information Systems, Inc.*
Steven B. Lipner	*The MITRE Corporation*
Teresa Lunt	*SRI International*
Frank Mayer	*Aerospace Corporation*
Dr. Catherine Meadows	*Naval Research Laboratory*
Sally Meglathery	*New York Stock Exchange*
William H. Murray	*Deloitte & Touche*
Noel Nazario	*National Institute of Standards and Technology*
Dr. Peter Neumann	*SRI International*
Nick Pantiuk	*Grumann Data Systems*
Donn Parker	*SRI International*
Dr. Charles Pfleeger	*Institute for Defense Analyses*
Professor Ravi Sandhu	*George Mason University*
Marvin Schaefer	*CTA, Inc.*
Daniel Schnackenberg	*Boeing Aerospace Corporation*
Miles Smid	*National Institute of Standards and Technology*
Brian Snow	*Department of Defense*
Dr. Dennis Steinauer	*National Institute of Standards and Technology*
Mario Tinto	*Department of Defense*
Eugene Troy	*National Institute of Standards and Technology*
Kenneth vanWyk	*Carnegie Mellon University*
Grant Wagner	*Department of Defense*
Major Glenn Watt, USAF	*USAF Strategic Air Command*
Wayne Weingaertner	*Department of Defense*
Howard Weiss	*SPARTA*
Roy Wood	*Department of Defense*

Awards Ceremony

6:00 p.m., Thursday, October 15
Convention Center, Terrace Level

A joint awards ceremony will be held at which the National Institute of Standards and Technology (NIST) and the National Computer Security Center (NCSC) will honor the vendors who have successfully developed products meeting the standards of the respective organizations.

The Computer Security Division at NIST provides validation services for vendors to use in testing devices for conformance to security standards defined in three Federal Information Processing Standards (FIPS): FIPS 46-1, *The Data Encryption Standard (DES)*; FIPS 113, *Computer Data Authentication*; and FIPS 171, *Key Management Using ANSI X9.17*.

Conformance to FIPS 46-1 is tested using the Monte Carlo test described in NBS Special Publication 500-20, *Validating the Correctness of Hardware Implementations of the NBS Data Encryption Standard* which requires performing eight million encryptions and four million decryptions.

Conformance to FIPS 113 and its American Standards Institute counterpart, ANSI X9.9, *Financial Institution Message Authentication (Wholesale)* is tested using an electronic bulletin board (EBB) test as specified in NBS Special Publication 500-156, *Message Authentication Code (MAC) Validation System: Requirements and Procedures*. The test consists of a series of challenges and responses in which the vendor is requested to either compute or verify an MAC using a specified randomly generated key.

Conformance to FIPS 171, which adopts ANSI X9.17, *Financial Institution Key Management (Wholesale)*, is also tested using an EBB as specified in a document entitled *NIST Key Management Validation System Point-to-Point (PTP) Requirements*.

The NCSC recognizes vendors who contribute to the availability of trusted products and thus expand the range of solutions from which customers may select to secure their data. The products are placed on the Evaluated Products List (EPL) following a successful evaluation against the *Trusted Computer Systems Evaluation Criteria* including its interpretations: *Trusted Database Interpretation*, *Trusted Network Interpretation*, and *Trusted Subsystem Interpretation*. Vendors who have completed the evaluation process will receive a formal certificate of completion from the Director, NCSC marking the addition to the EPL. In addition, vendors will receive honorable mention for being in the final stages of an evaluation as evidenced by transition into the Formal Evaluation phase or for placing a new release of a trusted product on the EPL by participation in the Ratings Maintenance Program. The success of the Trusted Product Evaluation Program is made possible by the commitment of the vendor community.

We congratulate all who have earned these awards.

15th National Computer Security Conference
Table of Contents
Refereed Papers

Tutorials [Track D, Room 301-303]

EXECUTIVE SUMMARIES

ACCREDITATION: IS IT A SECURITY REQUIREMENT OR A GOOD MANAGEMENT PRACTICE?

Thomas E. Anderson

USATREX International Inc.
7926 Jones Branch Drive, Suite 410
McLean, VA 22102-3303
TEAnderson@DOCKMASTER.NCSC.MIL

Abstract

This paper describes one of many possible concepts for accrediting an automated information system (AIS). Providing a contrast between certification and accreditation the reader will hopefully gain a better understanding of the accreditation process. Ideally, through good management and security practices, accreditation is accomplished before the commencement of system operations. This paper presents a process that could be used by Information System Security Officers (ISSOs) to accredit systems operating with outdated or no formal accreditation.

Keywords

accreditation certification configuration contingency defacto-accreditation evaluation risks safeguard threat

Introduction

Accreditation, evaluation, and certification: are they really one in the same or merely pieces of the overall puzzle? Who is responsible for each piece? Why do we care? These are some of the basic questions in discussion regarding security and the use of automated information systems. If you read the proliferation of government publications on accreditation and certification you will discover the lack of definitive guidance on the what, when, and how of accreditation and/or certification. In some instances the requirements for certification and accreditation even appear to be quite similar, if not the same. For the purposes of this paper I would like to define certification as simply the documentation and verification that security features, assurances, and safeguards are in place to protect the AIS. Furthermore, I would like to define accreditation as simply the managerial acceptance of the risks involved with operating the AIS in a given manner based on the certification evidence provided.

Every automated information system in operation today has been accredited by its respective organizational management. Some systems have been formally accredited. This accreditation has been documented in the form of an accreditation package and an accreditation statement signed off by an official of the organization. Others have what this author would refer to as a "defacto-accreditation". The day that management allowed the

system to begin and continue processing information there was an implied or defacto managerial acceptance of all the known risks involved with the operation of the system in the given environment.

Security Requirement vs Management Practice

Although most organizations consider accreditation a security requirement, it is really a culmination of good management and security practices. Through normal management practices the major portions of the documentation in support of an accreditation should exist for all AISs regardless of the sensitivity or classification of data to be processed. In most cases this documentation is utilized in the day to day management of the system. The following briefly depicts what I consider the major elements of this documentation to be; the depth and detail of this documentation are site and management dependent.

- Schematic drawings of the system showing peripheral devices, communications equipment, and all external interfaces. These types of drawings are usually found in the Operating Systems Maintenance and or the Computer Operations sections of an organization.

- A list of hardware components (corresponding to the schematic drawing above) and major software packages utilized. This listing should contain such information as manufacturer, model, serial number, generic device type, and location for each piece of hardware. Manufacturer, product name, generic product type, version, and release numbers should be included for each software package installed on the system. This type of information is the same as the information that should be used in the configuration management process for the system.

- Copies of all standard operating procedures (SOPs) which pertain to the use/security of the system. Containing step by step procedures, these SOPs should detail personal and organizational responsibilities. They should delineate the duties of each individual involved in the operation, maintenance, and security of the information system.

- Facility Risk/Threat Analysis. Without identification of potential risks or threats management has no way of determining if adequate precautions have been taken. There are many different ways to obtain a risk or threat analysis; some are quantitative and others are qualitative. The risk/threat analysis approach will vary from one organization to the next. There is no recommended best approach and the method used should be determined by the organization's need, policy, and management. The only stipulation is that a risk/threat assessment be performed, because a conscientious decision by an accreditation authority cannot be realistically made without some form of risk/threat analysis.

- Configuration Management Plan. The configuration management of an information system is paramount to its management and security. The introduction, removal, and or change to the components of the system (both hardware and software) must be strictly controlled. Strict adherence to the configuration management plan is essential to insure that security is provided to the overall information system.

- Contingency Plan. A complete and comprehensive contingency plan is an essential part of good management for any information system. This plan should cover every possible situation from worse case scenarios to minor disruptions. Responsibilities and actions to be taken should be clearly identified. Developing a good contingency plan is only half of the problem; the plan should be fully and functionally tested periodically. This testing should not be just a walk through, but an actual simulation of a disaster.

What About Trusted Products?

The National Computer Security Center (NCSC) evaluates the security controls of commercially produced general purpose operating systems for use by governmental departments and agencies using the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) [3]. The TCSEC, commonly called the "Orange Book", outlines criteria for evaluating the security controls or features built into an automated system. The criteria is divided into two basic types of requirements; security feature requirements and assurance requirements. Accomplishing the evaluation independent of future applications and irrespective of the physical environment of the hardware, the NCSC awards a level of trust rating. The level of trust rating for each system that successfully completes the evaluation process ranges from D (Minimal Protection) through A1 (Verified Design). "It must be understood that the completion of a formal product evaluation does not constitute certification or accreditation for the system to be used in any specific application environment." [3]

Certification

For each automated information system processing sensitive unclassified information within the federal government, Office of Management and Budget (OMB) Circular A-130 [2] requires application system certification. An official of each agency must certify that the security safeguards of the application system are adequate, have been tested, and meet all applicable policies, standards, and regulations. The safeguards do not have to be built into the hardware or software. Safeguards can be procedural methods, personnel security programs, risk/threat analysis, contingency plans, etc. Certification addresses the safeguards built into the hardware, coded into the software, and the other non-automated safeguards that enforce the security policy of the system. For non-automated safeguards, certification covers the existence of adequate and testable procedures, plans, and programs that meet all

applicable policies, standards, and regulations. Depending upon the mode of operation for the AIS these non-automated safeguards could be as important if not more important than the safeguards found within application software or the safeguard features built into the hardware. Since A-130 requirements for certification are quite similar to the requirements for accreditation of systems processing classified information, some organizations have taken a stance that sensitive unclassified AISs need to be accredited. Others believe that only the applications software that processes sensitive unclassified data needs certification.

Why Do We Accredit?

If the security controls of commercially produced general purpose operating systems are evaluated and assigned a level of trust, and the system applications and associated safeguards are certified to be adequate and meet all applicable policy, regulations and standards then where does accreditation fit in?

As previously stated, the NCSC evaluation does not take into consideration the application to be processed or the physical environment of the hardware. Certification, in most cases, only looks at the specific application or safeguard to ascertain its adequacy and compliance with policy, regulatory and standards requirements. In order to establish that it is truly acceptable to process sensitive unclassified and or classified information on an automated information system, management must be willing to accept the risks of operating the system in a given environment using established procedures and safeguards. This is where accreditation comes into play.

Accreditation is the final and most significant piece to the computer security puzzle. This is the first time that the hardware and operating system, with specific applications, in a given physical environment, using established procedures and safeguards based on identified risks, are considered from a security point of view. Using the certifications and their associated certification evidence to support the accreditation request, management must make a conscientious decision whether or not to allow the system to process at the requested level.

Computer systems operate in various configurations, such as stand-alone personal computers (PCs), local area networks (LANs), wide area networks (WANs), mini-computer and mainframe systems processing information ranging from unclassified to highly classified. Many of these systems may be formally accredited by either a Designated Approving Authority (DAA) or a Designated Senior Official (DSO) of a governmental organization, but others may be operating with outdated accreditation or simply no accreditation consideration at all (defacto accreditation). We find the PC system is usually put into operation without regard for accreditation while the mini computer and mainframe system may have been accredited at one point in time, but most likely the accreditation has not been updated or maintained with any

regularity. Local area and wide area networks pose many accreditors with the problem of establishing the scope or bounds of an accreditation.

Accreditation Process

Ideally, the accreditation process begins at the system concept development stage with formal accreditation being accomplished prior to the actual commencement of system operation. An accreditation plan should be developed at the beginning of the system life cycle and at individual system development milestones certifications and certification evidence should be gathered to form the basis of an accreditation package. The depth and detail of information necessary for an accreditation package should be commensurate with the sensitivity and or classification level at which a system is to be accredited. The accreditation of computer systems, whether they are already in operation or newly installed, requires tremendous cooperation and coordination between the Information System Security Officer and all involved in the development, fielding, and use of the information system. Having stated the ideal situation where the accreditation process begins at the concept development stage, the remainder of the process (Figure 1) outlined in this paper will deal with the real world situation facing many Information System Security Officers; the use of information systems that are operating unaccredited/defacto accredited or with outdated/expired accreditation. The documentary requirements are the same for both new installations and existing systems.

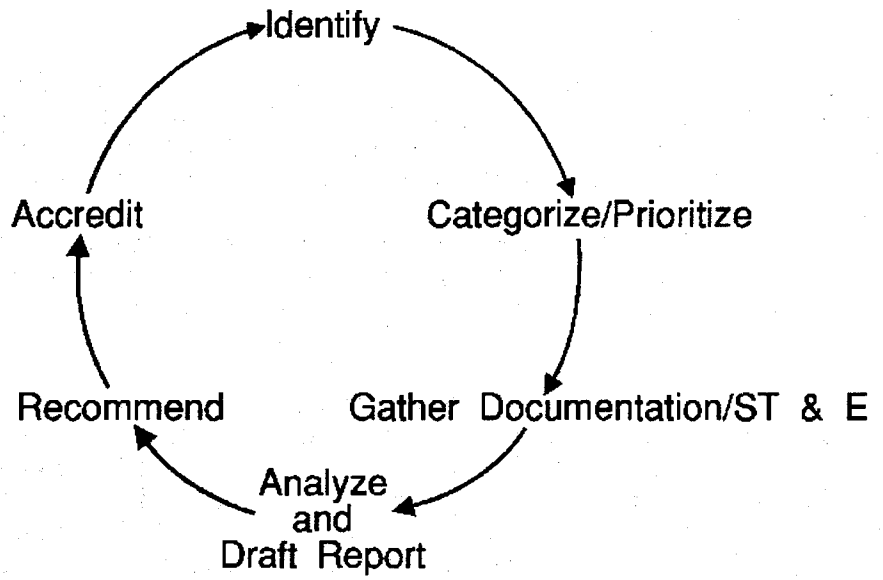


Figure 1. Proposed Process for Accreditation

Identify

Identification of the computer systems requiring accreditation is the first step to the successfully accreditation of an

organizations computer systems. The basic approach to this task is to go through the organization and physically identify each computer system and its configuration. The primary items of interest for this data gathering effort are equipment type, location, classification level and extent of existing accreditation.

Categorize/Prioritize

Once each system has been identified it should be categorized (i.e. PC, LAN, mini/mainframe, etc) and each of these categories should be further broken down by sensitivity/classification level of data to be processed. Once all of the organization's computer systems have been identified and categorized, the Information System Security Officer should assign a priority to each system and category for an accreditation team to prepare accreditation documentation. The accreditation team should consist of technically qualified individuals who can rapidly grasp the concept of operation and independently assess the information system's compliance with the organization's security requirements/needs.

Gather Documentation/System Test and Evaluation

Using the prioritized list the accreditation team should physically survey each system. The team should meet with the functional manager controlling the AIS to gain a better understanding of the concept of operations for each system and collect the documentation to compile a security profile (e.g., copies of standard operating procedures (SOPs), facility risk/threat analysis, any plans associated with the computer system (Contingency, Configuration Management, etc), and any previous survey, inspection or accreditation documentation). Depending on the mode of operation (i.e., dedicated, system high, etc) the team may need to perform system and security testing (System Test and Evaluation (ST&E)). Testing for those systems already in place would serve a dual purpose: 1) to verify that system security features exist and function properly; and 2) to verify that these security features have been properly implemented.

Analyze and Draft Report

After careful analysis of the information gathered by the team a draft accreditation report should be compiled. The report should identify the security level and mode of operation, note key vulnerabilities (both those identified by physical inspection and those identified through system testing), outline any exceptional circumstances pertaining to the operation at the requested security level, include a discussion on actions taken to reduce the risks, and provide justification for why these risks should be accepted. The report should also describe the concept of operation, providing information concerning features of operational and security modes. This would include descriptions of hardware, software, significant applications, interfaces, user population, and a description of features and or procedures from the various security disciplines

which support the operation. Attachments to the accreditation report should include the certifications and certification evidence such as the security profile consisting of standard operating procedures, associated plans, system and security test results, risk/threat analysis, and any previous survey/inspection reports (e.g. Inspector General reports, etc.).

To support the accreditation, some organizations might require such additional items as duty appointments for all system security related personnel, security checklists, non-security related plans associated with the system (e.g., training plan), etc. The types of documentation and the level of detail of each attachment to the accreditation report will be site/organization dependent. Each organization will need to use the sensitivity of the data to be processed and the complexity of the information system as their rule and guide for accreditation documentation requirements.

The draft accreditation report should now be coordinated with the functional manager to ensure the correctness of the concept of operations and the team's understanding of the security posture of the system. The Information System Security Officer, in conjunction with the accreditation team and the functional manager, should review and assess each report to finalize an accreditation package (the accreditation report with attachments).

Recommend

Based on the accreditation report and its attachments, the Information System Security Officer will take one of three possible actions:

1. Forward accreditation report to the Designated Accrediting Authority or Designated Senior Official with a recommendation for full accreditation;

2. Forward accreditation report to the Designated Accrediting Authority or Designated Senior Official with recommendation for an interim accreditation pending the resolution of identified deficiencies; or

3. Return the accreditation package to the functional manager with a list of operational and or security related deficiencies that require correction prior to the accreditation of the system.

The accreditation package should be treated as a living document with elements continually updated as the system evolves. It is recommended that a cycle be established for the periodic review of the systems currently accredited. The frequency of this review should be determined by management, based on the sensitivity of the information to be processed and the magnitude of change to the existing system.

Conclusion

Every AIS in operation today has been accredited, either defacto or formally and conscientiously, because ultimately someone in management has accepted the risks of operating the AIS in its current environment. Hopefully, if that someone is you, you are now asking yourself "How was my system accredited; defacto or formally"?

Acknowledgements

For their comments and suggestions during the preparation of this paper, the author would like to express his thanks and appreciation to William Hawkins and Michael Fuksa.

References

[1] Department of the Army, Security: Information Systems Security, 1 August 1990, Army Regulation No. 380-19

[2] Office of Management and Budget, Management of Federal Information Resources, December 1985, OMB Circular No. A-130

[3] Department of Defense, Department of Defense Trusted Computer Security Evaluation Criteria, 15 August 1983. Department of Defense 5200.28-STD

[4] National Computer Security Center, Glossary of Computer Security Terms, 21 October 1988, NCSC-TG-004, Version 1

[5] National Computer Security Center, Information System Security Officer (ISSO) Guideline, 1 November 1990, NCSC-TG-0??, Version-1 DRAFT

[6] U.S. Department of Commerce/National Bureau of Standards, Guideline for Computer Security Certification and Accreditation, 27 September 1983, FIPS PUB 102

APPLICATION LAYER SECURITY REQUIREMENTS OF A MEDICAL INFORMATION SYSTEM

Deborah Hamilton
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, California 94304-1126

Abstract

While a lot of effort has gone into evaluating the security needs of DOD applications, many commercial applications have not been sufficiently evaluated. This paper discusses the application level security requirements of a commercial application in part to stress the differences between the TCSEC requirements and some commercial application needs.

There are several major distinctions between typical DOD type systems and the commercial application evaluated here, a medical information system. Foremost is the importance of having access to the data when necessary even at the cost of the confidentiality of the data. Access controls must be strict on medical records; however, the medical team must be able to bend the rules in emergency situations. These emergency situations must still be carefully and securely monitored. The second major distinction is the difference between the power and administrative setup. A medical system can not be modeled hierarchically; it is more closely represented by a lattice structure. The third distinction is the importance of maintaining all data such that it can be admissible as court evidence.

The objective of this paper is to emphasize the need for more research on non-DOD security as well as to highlight important but yet unsolved and interesting research topics. This paper discusses the basic application level security concerns for a computerized medical information system and analyzes the requirements of the security concerns. It concludes by summarizing the most important requirements.

Introduction

Our health care system is at risk and desperately requires improvement. According to a recent government study by the Institute of Medicine [3], computerization of the health care system will provide better and more efficient care while cutting costs. The Institute of Medicine [5] documented in their study that medical records were unavailable in up to 30 percent of patient visits. Medical records may be unavailable because they have been misplaced, they have not been brought over from the patient's previous health care providers or there was not enough time to physically retrieve the records. This forces physicians to rerun tests and prevents dangerous trends from being spotted. Availability of medical data is especially important during emergency situations regardless of whether the patient has established a history at the particular institution.

Medical care is improved and costs are lowered when medical information can conveniently be shared among health care team members, researchers, accountants, administrators, health regulators and in-

surance groups. Much time and energy can be saved if paperwork between the above mentioned groups is minimized and information is exchanged electronically. [1] Physicians benefit from being able to consult with specialists and share patient data. Research groups benefit greatly from anonymous patient information. Physicians need access to networks for on-line databases with medical information and news groups to keep in touch with the many new medical developments. Ideally, a medical system would allow physicians to share information with specialists, compare patient data with diagnostic information, link patient data with family health history, and access the most recent research information. In addition, the amount of information physicians manually sort through in order to complete a thorough diagnostics job has become unmanageable without computer tools. [2]

The government and some major insurance companies have already made several proposals to allow electronic transfer of patient data among health care providers and insurance companies thus reducing paperwork and preventing duplicate testing. The Health and Human Services department has proposed both a national database for patient health data and a nationwide electronic billing system. The first proposal involves patients carrying a smart card which allows access to a centralized database containing patient records. The most recent proposal involves patients carrying a 'Credit Card' containing their entire medical history which would then be used to link into the insurance company and provide immediate notification of insurance coverage [3]. There are hopes that some part of an electronic billing system may be in place within a year or two.

Computerized medical systems are necessary and are quickly emerging; however, some important issues remained unsolved. Security is one such issue. The success of systems may ultimately hinge on the security aspects since "one catastrophic incident involving a computer-based patient record system could set the legal status of computer-based record systems back decades"[10]. Since security is an important but largely unevaluated aspect of medical applications, we have chosen to identify and evaluate the security requirements of a medical information system similar to but more sophisticated than the model being proposed by the government.

This paper is based on information from various sources: transcripts from several interviews with nurses, doctors and other staff in hospitals and clinics; documents on the legal regulations of patient data [4]; academic research; [11] industry [10] [7]; and government studies [3].

Scope

The number of security issues associated with security for medical information applications is vast. The scope of this paper is limited to application level concerns such as data integrity, non-repudiation, confidentiality, authentication, auditing, and access control. Other very important issues are not discussed here but the intent is not to deemphasize them. These issues include fault tolerance, recovery mechanisms, secure operating systems, secure networking, secure databases, security policies, politics, reluctance to using or trusting computers, password generation, training, ease of use, system maintenance and administration, viruses, worms, secure backups, secure data storage, secure hardware, human entry mistakes and quality assurance.

[1] Some physicians estimate that approximately 45% of their time is spent on paper work for insurance companies.

[2] Refer to *The Computer-Based Patient Record* [3] for a more in-depth discussion of the need to computerize medical systems.

[3] See the San Francisco Chronicle - June 22, 1992 - Front page - *U.S. Medical 'Credit Card' Proposed*

[4] See appendix B of *The Computer-Based Patient Record* [3]

Security Concerns

A computerized medical system has many strenuous and complex constraints and requirements. An application used in medical diagnosis and record maintenance must work flawlessly because of the lives that depend on it. The consequences of inadequate security can be life-threatening or financially devastating to the health care group: a person could be given improper medical treatment or refused treatment; a lawsuit could result if security is breached or data integrity threatened.

Security in medical applications is necessary for three reasons: to prevent bad medical care; to prevent abuse such as unauthorized access to information or prescription drugs; and to provide accountable records for malpractice cases. Incorrect patient data could result in bad medical care or could prevent one from getting health insurance. Information leakage of highly subjective diagnoses could prevent someone from getting insurance or a job. This problem will only get worse as we are able to gather more information on patients. For example, the ability to read genes might allow physicians to determine whether a patient is highly susceptible to alcoholism, colon cancer, Alzheimer's or diabetes. Such information is helpful in the right hands but very dangerous in the wrong hands.

Medical applications have one very important requirement that other fields typically do not. A life may hinge on the ability to get access to all of the information at any time. Many people in the medical field believe that access to medical information should not be severely restricted but rather carefully audited. As one paper describing a medical application put it "We argue that the single, most important success factor of this project is in providing immediate convenient access to patient's clinical information, whenever needed, from anywhere... [7]".

Current paper record systems are carefully regulated with complex and constantly changing rules to provide safety and confidentiality and to ensure admissibility into court in case of a lawsuit. These rules, however, must be carefully re-evaluated for computerized systems. When medical information is made available from a computer application, a breach of security becomes more tempting due to the perceived ease of access to massive amounts of patient information and the perceived anonymity. Leaks in security and mistakes in data integrity become even more devastating in a computerized system since humans may not always be available to filter or review the data.

Requirements and Analysis

This section describes the requirements of current non-computerized medical information systems and analyzes the security requirements of future computerized systems.

Data Integrity and Non-repudiation

Data integrity is of the utmost importance in a medical system. For a medical application to succeed, there must be some mechanism which insures both highly reliable and verifiable data during storage, transmission and display. Data integrity mechanisms should be available for many forms of data including text, images, voice, and video.

In any successful medical application there must be a mechanism to achieve the same reliability as written signatures and with the same ease. A medical system must have a very good mechanism for verifying who has entered, agreed to, or ordered what. Whether a physician has entered notes or dictated them to someone else, as often occurs, the physician's written signature is an effective method of verifying that information has been entered on their behalf or at least that they have read and agreed

to the information thereby taking responsibility. There must be a mechanism to sign for another's entry but still retain information on the original enterer of the data. Signatures are also necessary to show others that the physician gives permission, for example, to dispense a prescription. Patient signatures are necessary, as well, to indicate that the patient has read and understand information and/or given permission for a procedure.

Ideally, the signature mechanism would work for both the medical staff and the patients. If a patient's permission can not be stored directly in the computer, hard copies of patient consent forms will have to be stored separately from the records or scanned in - possibly resulting in lost forms or too little resolution for verification of the signature. To store the patient's permission on-line, there must be an simple but effective method of producing and entering secret keys. These keys could be derived from patient passwords, smart cards or biometric mechanisms. However, they must not be passive forms of entry such as a digital fingerprint which can be taken while asleep, rather the patient must be awake and fully aware that they are giving permission.

Signatures must be available independent of whether the information is in clear-text or encrypted and must be completely tamper-proof in order to be admissible in court. A secure date and time stamp should also be retrievable on the signature if possible. In addition, there must be a mechanism to ensure that the signature is interwoven with the appropriate information to ensure that the signature is not reused or separated. Those with only read permission should be able to read the signatures associated with the data. The signatures should be decryptable for a long time - hopefully for the life of the information [5].

Medical systems require a mechanism to ensure that a request is submitted once and only once. For example, the following abuses must be prohibited: a patient copying an order for a prescription and resending it at a later date to obtain refills without authorization; a pharmacist copying prescriptions to resubmit multiple times thus making records account for missing drugs.

In order for medical records to hold up in court, records must be kept up-to-date and contain the name, time and date of any changes or additions to the records. Deletions must be logged as well and most "deleted information" saved [6]. Regulations stipulate that the records remain easily modifiable and that the most up-to-date information is easily identifiable. Copies of records must be trustworthy so creating, accessing and storing records must be tightly controlled.

Authentication

A secure authentication mechanism is very important in a medical information application. The mechanism must be simple and easy to use since the health care team is typically uncomfortable and unfamiliar with computers. Since individual accountability is a must, individual ids are be required. The authentication mechanism should not falsely deny authorized users.

To provide secure authentication, users must log off immediately after relinquishing physical control over the keyboard. Not logging out results in an authentication problem since the system is unable to ensure that the keyboard has not switched hands. Medical professionals usually resist logging off systems immediately after relinquishing physical control over the key board unless login is easy and painless, startup quick and the physical state of the user workspace retained when logging out. Health care providers are frequently interrupted and will not think about logging out when called for an emergency.

[5] One administrator of a hospital that plans to go completely to electronic storage of patient data in four months indicated that they were planning on using digital signatures encrypted with the user's passwords. These passwords must be changed every six weeks by the users and there are no plans to store the old passwords. This mechanism will not stand up to regulations nor will it allow records to be admissible in court.

[6] Regulations stipulate which information can actually be deleted.

Since authentication is such a vital problem in medical applications, a mechanism for automatic logout should be considered. However, a simple automatic logout mechanism which logs users out after n seconds or minutes of not using the machine would be insufficient since a small n is very inconvenient and will cause users to find methods to bypass security and a large n creates security problems. It is important to find a method of letting the system know when the users stray from the computer.

Possible changes in legislation may soon grant patients greater access to medical records in order to check for inaccuracies. If so then ideally patients would also be able to use the authentication mechanism. This would require a mechanism that can handle a very large number of users especially in a centralized system where ideally every man, women and child would have a record on the system.

Auditing

Many people in the medical profession believe that it is highly preferable to audit access and actions rather than to severely curtail these activities. A medical information application should have a very secure and reliable audit system to detect abuses and problems. The level of auditing should be variable based on the application, the user, and the mode of operation (e.g. normal, emergency). Secure auditing is needed to make sure that abuses are detected. The audit trail should detect unauthorized reads and modifications, malfunctions, and corruptions. Secure auditing is also needed for tracking down inaccurate data. For example, if a lab test result is entered incorrectly and later corrected, there must be a mechanism to determine where the incorrect data was used and who must be notified of the correction. Otherwise, future treatment may be based on incorrect information. Auditing to detect exploitation of covert channels, inference and aggregation attacks would also be helpful.

Auditing should take place on both record and application levels. For example, an audit trail must be kept of who accessed which files and who is currently looking at them, as well as who has ordered which tests and which prescriptions and who has modified what information within the patient record. The audit trail must also keep track of who has forwarded what information and to whom. It is important to collect and correlate audit data from a number of different levels, stages and abstractions for the information to be meaningful. This will determine clues such as whether the user knew what he was looking for. For example, did the user know what keyword should be used in a search or did they guess multiple times before coming across a correct one. There must be a method, however, of eliminating superfluous or misleading information to prevent excessive record keeping at the same time as retaining the essential information necessary as evidence in possible court cases.

Access Control and Confidentiality

Medical applications have very demanding access control needs. Access controls should be dynamic and flexible yet strictly regulated and operated close to the least privilege principle when possible. However, the most important characteristic is that no authorized person should ever be refused access when needed especially in an emergency situation. In addition, even unauthorized personal might need to have access to patient records under emergency situations. It is not always possible to find a person able to give access permission in an emergency situation. Therefore, it is better to allow access in emergency situations and review the situation afterwards than to deny access. The government's proposals discussed at the beginning of this document allow patient data access to be set up in two ways: one could allow access only to those that have the patient's card or one could allow access of the patient database to everyone who has access to the system. A solution in the middle would be better. For example, access is given to a predefined list of health care providers for each patient. In addition, a predefined set of users are allowed to access any patient's records when they declare an "emergency"

mode. When they enter this mode, however, all of their actions are monitored and evaluated at a later time to prevent abuse.

Users should be able access a medical application anywhere in the local network and possibly off-site as well. One medical application's development team noted that physicians welcomed their computer application especially since they could now access information from home. Security of remote access, however, was not discussed [9].

Medical systems are not accurately represented in hierarchical fashion. Not only are there many different types of health care workers but their rights are not hierarchical; many rights overlap. The model of authority is most accurately modeled with lattice structure where for example, physicians have a lot of power over modifications to patient records but have little power or no power over auditing controls. System administrators have power to modify the auditing controls but yet have no power to modify patient records. The following list illustrates the complexity of access to users. Described below are some examples of people who may need access to various patient data information: [7]

- Physicians - for background information; to keep track of patient notes, current status, diagnosis and treatment; for literature searches; for consulting with other physicians both publicly (notes groups) and privately (electronic mail); for access to on-line medical databases; for comparison of current patient's symptoms with other patients' symptoms; to provide links into decision support systems [8]
- Nurses - for information on patient preparation to be done (e.g. blood pressure check); for recording patient statistics and relevant information
- Clerical staff - for appointment management; for hospital admissions; for maintaining patient information such as addresses; for registering patients
- Technicians - for information on specific tests to be performed; to enter test results
- Computer administrators - to make sure the system is running properly; to fix problems
- Hospital administrators - to determine statistics on patient load, efficiency, number of referals etc. to be used in evaluating and improving quality assurance in the hospital or clinic; to allocate resources; to develop and manage budgets
- Accountants - for billing purposes
- Insurance agents - to pay clients and providers; to check the validity of claims
- Researchers - to gather clinical information for studies [9]
- Social workers - to flag possibly suicidal patients; to determine possible abuse cases
- Pharmacists - for drug information; for prescription information; to determine possible side-effects and complications
- Mental health care providers - to store data on medications; to check for possible complications

[7] In this paper specific labels have been used for medical staff such as nurse, however, job duties are hardly ever broken down this cleanly. One person may perform some "nurse duties" and some "clerical duties". Job duties and titles vary immensely among the various clinics and hospitals.

[8] On-line information used in conjunction to decision support systems are already in use in some emergency rooms. They have been so successful that some insurance companies give a 20% decrease in malpractice premiums to those that use it in Massachusetts.

[9] The need for correlated anonymous patient data is expanding continuously. Medical care will improve at a faster rate once patient data can be compared electronically. Less money will be needed for studies if on-line patient data can be used for the first stage studies.

- Dentist - to determine drug allergies; to determine possible complications
- Patient - to review patient data for accuracy [10]

The above list indicates that access cannot be represented by an hierarchical structure. Many different people need access to different pieces of the records but no one should have access and modification rights to all the pieces. Access, addition, modification and deletion rights must be separately assignable. For example, pharmacists may need access to a patient record to check for interactions between medications as well as to add notes to the medication section of the patient records. However, they should not be able to change other parts of the record. In addition, regulations require that deleting information be permitted only under very strict situations and thus must be more tightly controlled than the other rights. It is critical that massive copying, searching and modifying of patient records be very tightly controlled to ensure patient confidentiality.

Some portions of the patient record are especially important to secure, for example, HIV-antibody test results, records of drug and alcohol abuse, psychiatric records, and records of celebrity patients. In addition, private information may be given in confidence to the physician in order to aid patient care such as sexual preference or abortion history. Some portions, however, must be more openly available. A system should be able to, for example, allow easy access by any health care worker to notes on whether a patient's bodily fluids require special precautions. Also, treatments should be available to billing groups and insurance companies so that information may be shared or at least forwarded.

In addition to the many different types of access rights, the access relationships must be very flexible and dynamic. Ideally, it should be possible to configure the system to regulate access based on any of the following:

- Job (i.e. physician, internist, subinternist, chief resident, nurse, technician, accountant, security officer [11])
- Relationship to patient (i.e. primary or consulting physician)
- Area of specialization (i.e. pediatrics, internal, radiology, dietitian, intensive care)
- Patient status (i.e. inpatient versus outpatient, under-treatment)
- Individual (i.e. one nurse may need access to a particular set of patient's data)

A flexible role-based access mechanism is important. It would be difficult to shift capabilities in a medical clinic without a role-based system. Nurses are asked to support particular doctors but may be reassigned frequently. Primary physician roles are frequently changed which would entail only one change in a role-based system and possibly many on a capability-based system depending on the organization. It would be tempting in a capability-based system to assign someone to a job category and allow the maximum privilege to that job to ensure access when needed. Since many of the responsibilities overlap between job categories, maximum privilege for each one could leave access wide open. It may be sufficient for all physicians to have access to all patient data records, for example, if their access is monitored sufficiently. However, this would entail much more auditing and monitoring than would be necessary with a better suited access control mechanism.

As an additional benefit of a role-based system, the access control maintenance responsibilities can be divided among the central and local applications. The central application could define the rules

[10] Some states allow patients access to their own record to check for inaccuracies.

[11] As previously mentioned, titles and duties vary so this mechanism must be flexible. One person may fit into various categories.

(e.g. the access rights of primary physicians) while leaving the role definitions (e.g. which individual is actually tagged as the primary physician) to the individual medical centers. This frees the central system of some details and leaves some flexibility to the local health care center (eg. to define an emergency mode). This simplifies access control management but forces more trust to be placed on the local health care system.

It would be useful to have a two-party permission system for some situations. This might help prevent mismanagement or fraud such as prescribing tests which are either unnecessary, questionable or not given. It would also allow tighter control over certain critical operations such as modifying secret keys and access rights.

A delegation mechanism for one person to temporarily or permanently sign over control to another would be useful. This would allow primary physicians who go on vacation, for example, to sign over primary physician's responsibility to another. However, a delegation mechanism is not as essential with mechanisms in place that allow protected and monitored emergency access.

There should be a mechanism for allowing patients to give "permission" for others to access their files for longer than a single login session. For example, if a smart card is required for access, it would be very inconvenient to require that the patient either leave the card or come back in two days when the blood test results are back from the lab and ready to be entered into the patient's record. This extended permission should, however, expire after a set amount of time.

Labels indicating the sensitivity of information may useful in this type of application but are not essential. Health care professionals have been trusted in the past to know what information is confidential and it is usually obvious who has a need-to-know with medical information. However, labels could serve as a reminder.

Unmodifyable labels identifying the origin of information might be somewhat useful but certainly not essential. It would not be useful in determining a security leak when the information is simple enough to forward without copying, such as identification of an AIDs victim. However, it would be useful in some instances such as when an insurance company is caught storing information that was illegally gathered.

Communication Over Networks

A medical information system, especially a centralized database system such as the one purposed by the government, has some very important communication requirements. A medical application must be able to assume that there is a secure network messaging mechanism to maintain confidentiality and integrity during transmission especially over unsecured lines. Whether the encryption should be end-to-end or link would depend on the structure of the network. End-to-end would prevent the end links from having to trust intermediate nodes. If the system is set up to provide access based on patients (as when using patient smart cards) and there is one centralized system such as the government has proposed, end-to-end encryption would make sense. Link encryption would necessitate a more complicated auditing mechanism and would require trusting the intermediate links.

Two-way trust is essential in a medical system. The receiver must be able to trust the integrity and authenticity of the sender and vice versa. A large portion of the communications will require confidentiality so a fast encryption mechanism is important. There should also be a mechanism for preventing replays and misroutings.

Summary

Medical information systems are just one example of a commercial application with interesting and challenging security problems. Some of the most important application level security concerns of this type of system are still open research topics. Others are technologically feasible but have not been implemented. Still others have been implemented but not yet integrated into large systems or are not commercially available and supported. There is much work to be done at all levels [12].

The most important security requirements of medical information systems at the application level are: integrity checks, secure and intelligently coordinated auditing, emergency access, secure identification, automatic logout, electronic signatures, secure communications, and role-based access controls. Secure and intelligently coordinated auditing, emergency access, and role-based access control mechanisms require extensive research before medical applications using them can be effectively implemented. These areas seem to receive less attention because they are not as important in DOD type applications.

References

[1] Barclay, M.L., B.L. Shipman, and S.F. Grefsheim *Implementation and Use of a University-based Wide Area Network for Access to the Medline Database*, Annual Symposium on Computer Applications in Medical Care, November 1990, p. 380.

[2] Brannigan, V. and B. Beier, *Standards for Privacy in Medical Information Systems: A Techno-Legal Revolution*, Annual Symposium on Computer Applications in Medical Care, November 1990, p. 266.

[3] *The Computer-Based Patient Record: An Essential Technology for Health Care*, Institute of Medicine, Washington, D.C. 1991.

[4] Hamilton, D., *Identification and Evaluation of the Security Requirements for Medical Applications*, Proceedings of IEEE Computer-Based Medical Systems Symposium, June 1992, p. 129.

[5] Lincoln, T. and Daniel Essin, M.D., *The Computer-Based Patient Record: Issues of Organization, Security and Confidentiality*, IFIP: Database Security Workshop IV, November, 1991.

[6] National Institute Of Standards And Technology, Gaithersburg, MD., *Minimum Security Functionality Requirements For Multi-User Operating Systems*, Draft - Issue 1, January 27, 1992.

[7] Ribitzky, R. et al., *Use of a Text Retrieval System to Automate Discharge Summaries and Operative Reports*, Annual Symposium on Computer Applications in Medical Care, November 1990, p. 370.

[8] *Secure DBMS Auditor: Final Technical Report and Functional Specification*, Trusted Information Systems, Inc., Contract Number F30602-87-dD0093, December 28, 1989.

[9] Teich, J.M. et al., *Design Considerations in the BWH Ambulatory Medical Record: Features for Maximum Acceptance by Clinicians*, Annual Symposium on Computer Applications in Medical Care, November 1990, p.735.

[10] Tang, P. et al., *Physician Workstations: Integrated Information Management for Clinicians*, Annual Symposium on Computer Applications in Medical Care, November 1991.

[11] Ting, T.C., *Application Information Security Semantics: A Case of Mental Health Delivery*, IFIP Database Security Conference, 1989.

[12] Refer to [4] for a brief comparison of current technology with these security requirements.

AN APPROACH FOR MULTILEVEL SECURITY (MLS) ACQUISITION

Bill Neugent
The MITRE Corporation
7525 Colshire Dr.
McLean, VA 22102, U.S.A.
703-883-6632

1. Introduction*

Lack of Multilevel Security (MLS) within United States (US) Department of Defense (DOD) computer systems is recognized as a significant shortcoming, because it limits interoperability and data fusion. To help address this problem, the Joint MLS Technology Insertion Program was officially established in January 1990. The program is managed by the Defense Information Systems Agency (DISA) and the security coordinator is the National Security Agency (NSA). The purpose of the program is to expedite the fielding of MLS operational capabilities within DOD. This paper is derived from guidance produced by the program [1].

This paper presents an approach for an MLS acquisition process for use over the next few years. This process is needed because of the great uncertainty and development risk currently associated with the development and acquisition of MLS capabilities. This uncertainty and development risk necessitate a flexible development and acquisition process and especially necessitate a process with less burdensome documentation than required in the current DOD software development standard [2]. This process is not intended for use by all sites -- only those with sufficient expertise and resources to deal with the complexities and difficulties currently associated with MLS. This process is intended as interim guidance, to be replaced within a few years by official DOD security acquisition guidance.

This is an idealized process rather than one to be inflexibly and uniformly applied to all sites. Furthermore, the process must be interpreted to best suit the particular people and organizations involved. The value of this generic MLS development and acquisition process is that it is a target that will improve development and acquisition effectiveness to the extent that it can be followed.

2. Activities

Figure 1 summarizes the phases involved in defining and fielding MLS capabilities. The three phases are (1) formulate and coordinate the approach, (2) acquire and integrate the capabilities, and (3) operate the system with the new capabilities. The following paragraphs examine the three phases in more detail.

---

* This paper is based on work performed under Contract DAAB07-91-C-N751 for the Defense Information Systems Agency (DISA).

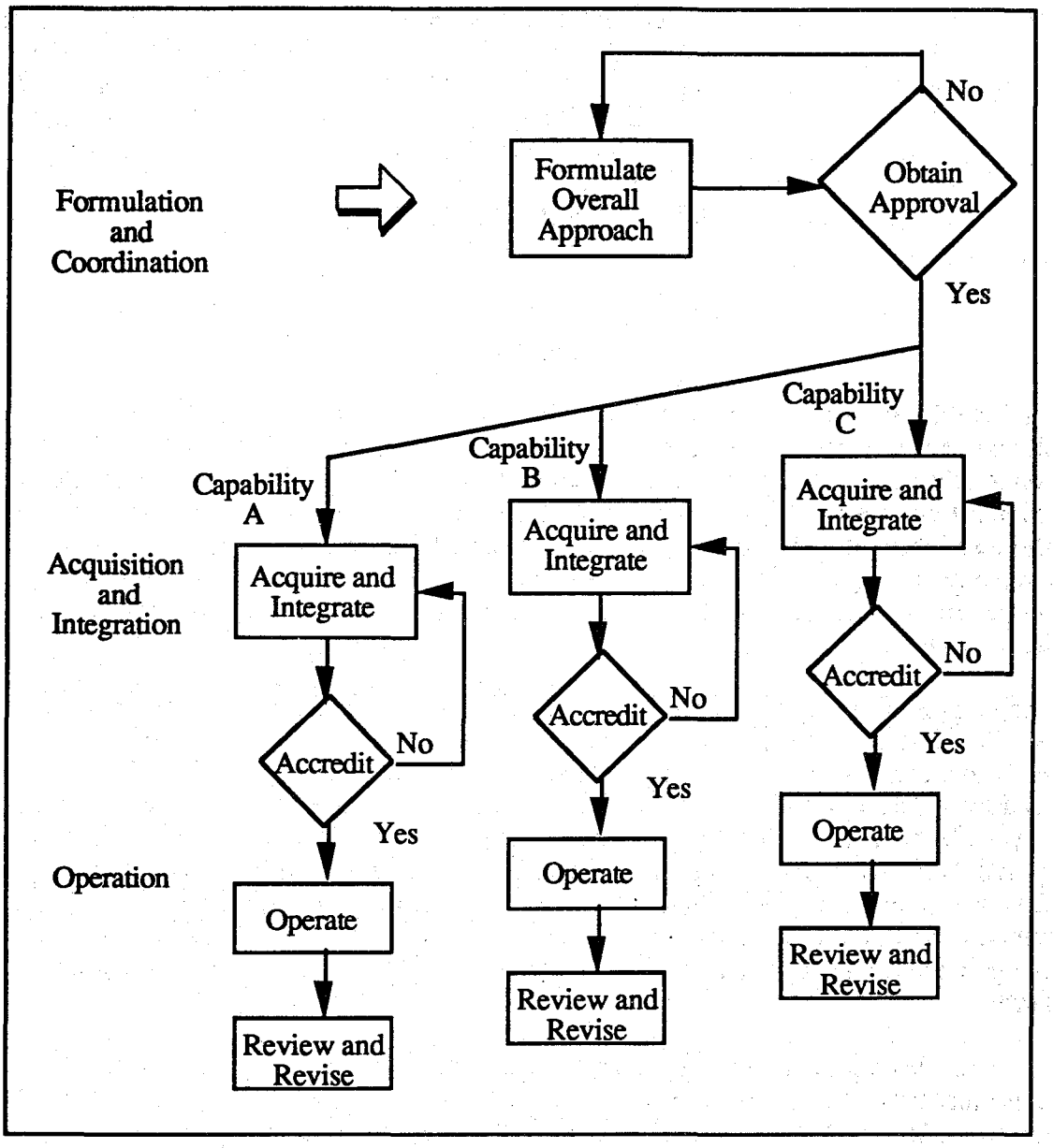


Figure 1. MLS Development and Acquisition Process

The formulation and coordination phase initiates activities and requires official approval. The approach that is formulated and coordinated in this phase incorporates all planned capabilities. Subsequently, during acquisition and integration, logically distinct capabilities are separately acquired and integrated. Separate acquisition approaches and schedules can be used for each capability, with each capability being independently integrated into the operational system(s) and the revised system certified and (re)accredited. Each capability then is placed into operation. To complete the process, the capability, the overall architecture, and the overall acquisition approaches are reviewed, with revisions incorporated into future iterations of the process.

This generic MLS development and acquisition process is partly based on ideas drawn from the Spiral development model, although the process as a whole is quite different from the Spiral model [3]. Taken from the Spiral model are (1) an emphasis on management of development risks, (2) use of prototypes, and (3) a streamlining of process, review, and documentation (in comparison with the DOD software development standard [2]).

2.1 Formulate and Coordinate Approach

The first phase of activity is to formulate and coordinate the overall approach. Five steps are involved in this phase:

- Select functional MLS capabilities
- Develop concept of operations briefing
- Obtain approval for approach
- Develop security architecture
- Develop acquisition and integration plan

While these steps call for several documents, the documentation must not be detailed or voluminous, since these characteristics would lead to inflexibility. Rather, the intent is that the documentation be sufficient to ensure adequate analysis is done to guide and plan the effort. Vugraph presentations should provide sufficient detail for the first four steps.

2.1.1 Select Functional MLS Capabilities

The first step in formulating and coordinating the approach is to select the functional MLS capabilities to be provided. This entails (1) assessing the availability of relevant, acceptably-mature trusted products, (2) identifying the major security threats and resultant risks, (3) defining the most critical operational needs, (4) identifying and complying with relevant security policies, and (5) reviewing available DOD architectural guidance.

While a general understanding of user requirements is assumed to exist, note that a detailed description of user requirements is not prepared in this or subsequent steps. The purpose of this generic process is not to identify and develop what users ideally would like to have, but to find, integrate, and adapt commercial trusted products that acceptably satisfy user needs. The emphasis thus is on commercially-available *approaches* and their acceptability rather than on refinement of *requirements*. It still is necessary to consider the concept of operations in selecting functional MLS capabilities, in order to ensure that the MLS solution addresses a legitimate need.

Due to the development risks currently involved with MLS, sites developing or acquiring MLS capabilities should adhere to the following criteria:

- Carefully scope and bound efforts so that risks are manageable; do not attempt to address too complex or too many MLS problems or products at the same time.
- Use products that comply with DOD standards for security, interoperability, or commonality.

- o Ensure that the approach is technically sound and noncontroversial and is not based on narrow assumptions about use or environment.

- o Ensure that product configuration management and maintenance are not undermined by adaptations or modifications.

- o Use sufficiently mature products to avoid wasting resources by assisting vendors in product debugging.

Once an initial approach is selected, an analysis is needed to ensure that equivalent operational capabilities cannot be provided through an approach involving less development risk. This analysis should examine required data flow and investigate the feasibility of alternate approaches, such as changing system operating levels or using a simple security guard.

2.1.2 Develop Concept of Operations Briefing

The second step is to develop a concept of operations briefing for the entire system, but with emphasis on MLS. The concept of operations should identify (1) specific data (and sensitivity levels) to be processed, (2) user capabilities (and clearances), (3) the system management approach, (4) the maintenance approach, and (5) the approach for evolutionary integration of new MLS capabilities with existing and planned operational systems. The concept of operations also should estimate short and long term costs, including any savings.

The concept of operations must explicitly address the man-machine interface, with emphasis on procedures that might be seen by users or system managers as being complex or cumbersome. It must also address functional limitations, such as a loss of particular capabilities or an inability to support particular types of commercial software. To counterbalance any such losses, the concept of operations also must explain MLS benefits, such as improved information access, improved data fusion, improved interoperability, reduced need for high clearances, and reduced amounts of hard copy output (to downgrade and handle).

One purpose of the concept of operations is to ensure that planners do not implement MLS for its own sake, but that they think through the implications of adding MLS -- both positive and negative. This helps ensure that the approach makes sense both technically and operationally. As part of the concept of operations briefing, a vugraph or two on the overall security architecture is needed for technical context.

2.1.3 Obtain Approval for Approach

The third step is to obtain approval for the approach. This involves coordinating the approach with local personnel, to ensure that benefits of the approach justify the acquisition and operational costs in the eyes of all involved people. Local personnel who should be involved include data owners, data users, accreditation authorities, system managers, security managers, system planners, local vendors, and local Independent Validation and Verification (IV&V) personnel. Necessary approvals must be obtained. This coordination can be time consuming and complex, but it is critical to the operational success of the capabilities produced by the process.

2.1.4 Develop Security Architecture

Once the approach has been approved, it must be expanded into a high-level security architecture that describes the approach in more detail. The architecture must identify involved system components and security functions and must identify the role each component serves in performing the security functions. Included in the architecture is the security policy for the system, which describes specific classifications, categories, and handling restrictions; identifies discretionary access control rules; and so forth. The security architecture, just as the concept of operations, must span all capabilities, even though a separate acquisition and integration effort is used for each capability. The reasons for this are to reduce the amount of official review and approval needed and to ensure integration across capabilities. The security architecture also must address the MLS capabilities in the context of the operational system(s) within which they are to be fielded.

2.1.5 Develop Acquisition and Integration Plan

The next step is to develop a plan for acquisition, integration, certification, and accreditation. The plan must clearly identify roles and responsibilities. This requires working with product vendors, program personnel, IV&V personnel, and other relevant organizations (e.g., procuring agencies for particular products) to identify needed hardware, software, integration, analysis, certification, accreditation authorities, and documentation.

Particular attention is placed on development risks, which are explicitly identified and prioritized in the plan and monitored during the effort. Development risks are critical areas warranting added resources or attention. Main potential development risk areas for MLS include integration, management, use, certification, and accreditation. Each of these potential development risk areas must be closely examined. For example, integration risks are examined by analyzing protocol, data format, security labeling, and interface standardization and compatibility and identifying any needed capabilities that are not available. This understanding of development risks is needed not only to identify where to focus attention, but also to plan the specific acquisition approach. For example, if a main risk area is the lack of well-defined user requirements, then the acquisition approach must ensure that integrators and developers work closely with users.

As part of the plan, the approach for product selection must be identified. Also to be identified is the detailed process for both technical and programmatic oversight, including assignment of official design authority and provision of means for team leaders to coordinate and resolve issues across product, application, and technical boundaries. The plan must summarize the process for moving capabilities from the prototype environment into the operational environment and must explicitly identify activities that are outside the scope of the prototype environment but are necessary for ensuring security in the operational system (e.g., planning for physical security, virus protection). The plan must identify the approach and resources (e.g., responsible organizations) for certification and accreditation. Finally, a determination must be made that adequate funds are available (or obtainable) to implement the plan.

2.2 Acquire and Integrate Capabilities

The second phase of activity is acquisition and integration. As shown in figure 1, separate acquisition and integration efforts are used for logically distinct capabilities, with each effort uniquely tailored to each capability. While some capabilities might take a year or two to develop, it is desirable that at least one capability be fielded within six months, so that immediate benefits can be seen. There are six steps in acquiring and integrating the MLS capabilities:

- o Acquire products
- o Develop and integrate the capabilities
- o Develop capability baseline
- o Perform functional testing
- o Perform certification
- o Support accreditation

2.2.1 Acquire Products

The first step is to acquire the products. This involves assessing and selecting the specific products to be used and then acquiring the hardware and software. Care is needed to ensure adequate competition among qualified vendors and to ensure that security issues are adequately addressed in the acquisition package. Guidance is provided by Abrams, et al. [4] and by Caddick [5].

2.2.2 Develop and Integrate the Capabilities

The major step in this phase is to perform development and integration. Within this step are the most pronounced differences between the multiple acquisition efforts. Several different development approaches are illustrated in figure 2. The distinguishing factor of each approach is the type of prototype implemented. The determining factor in deciding which approach to follow is the nature of the development risks involved. All approaches begin with refinement of the concept of operations and security architecture.

Where the main development risk areas are the user requirements and user interface, a **demonstration prototype** is needed. A demonstration prototype allows users to experience the look and feel of screens, menus, and reports. Based on this experience, requirements are redefined, new requirements generated, and possibly a revised demonstration prototype developed. If the main development risk area is the security management interface, a demonstration prototype still is applicable.

Where the main development risk area is technical integration, a **design assessment prototype** is needed. A design assessment prototype allows designers to examine technical integration issues such as protocol interoperation and commercial software compatibility, as well as issues such as platform performance, optimization techniques, and portability to target systems.

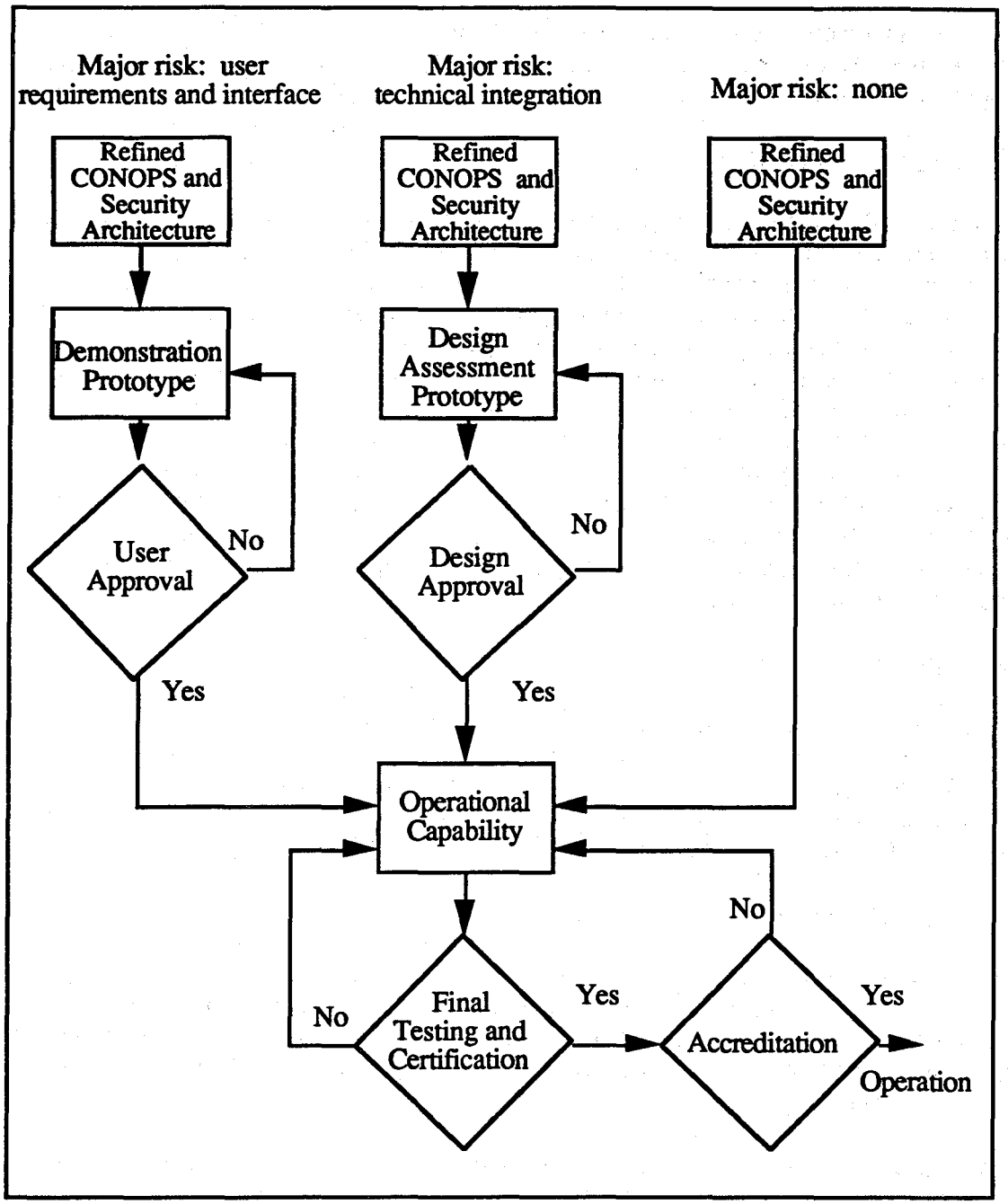


Figure 2. Development Approaches

Where there are no overriding development risk areas or where other prototypes already have been used to mitigate risks, an **operational capability** is developed. On completion of the operational capability, it is integrated into the operational system(s), testing and certification are completed, and accreditation is performed. New releases of the operational capability occur as needed.

Whatever approach is chosen, staff personnel must be trained in the approach and related technologies. To the extent feasible, prototypes should result in immediate operational benefits, in that some subset of the prototype should be suitable for near-term fielding.

2.2.3 Develop Capability Baseline

In parallel with the preceding step, a capability baseline should be prepared that serves as the official functional baseline for each capability. If the capability is based primarily on commercially-available elements, preparation of the capability baseline will involve assembling the refined concept of operations and security architecture, along with the available product documentation, and drafting any other documentation needed to reflect an integrated view of the involved products or components and the system into which they are to be incorporated. If substantial software development is involved, a capability baseline must be prepared to describe the new capabilities. Minimum contents for the baseline include functional capabilities, internal interfaces (e.g., data, other products, support and management software), and performance goals. If the baselined document is affected by user-required changes, the changes should be tracked and the document updated after software delivery. Demonstration prototypes can be a particularly important source of insight for the capability baseline.

2.2.4 Perform Functional Testing

The next step is functional acceptance testing. Where feasible, functional testing is done in the prototype environment, so as not to interfere with the operational system. Final functional testing typically is done after the capability has been integrated into the operational system(s). Functional acceptance testing should be based upon test scenarios in the capability baseline, as adapted for the specific site involved.

2.2.5 Perform Certification

Certification is the technical assessment of whether a system meets its security requirements [6]. Certification is performed in parallel with development and integration and is not restricted to testing at the end of the effort [7]. For example, early certification review is needed to prevent planners from pursuing approaches that have substantial security shortcomings.

At a minimum, all security-relevant documentation must be reviewed by certifiers and independent testing must be performed, including penetration testing. That is, if functional testing is performed by the developer, key portions must be repeated by the government (e.g., perhaps through an IV&V organization). Penetration testing must be performed by a different group from the one performing functional testing (preferably by a different organization) and must not be required to repeat the systematic, thorough examination of capabilities that is performed by functional testing. Penetration testing instead must be free to concentrate on arcane attacks and on areas of potential vulnerability. Data integrity and denial of service attacks are within the scope of penetration testing. Note that substantial testing is warranted due to the inherent security risks of fielding MLS capabilities and using new trusted products. Use of products rated or endorsed by NSA is expected to reduce, but not eliminate, the need for certification review.

The most critical aspect of security certification is the use of qualified specialists to perform the work. Each security product involved must be examined by an objective expert who is qualified to assess the product's security effectiveness within the particular capability. For example, if NSA-evaluated products are used, a representative from the product evaluation team could participate in the certification. Without such specialized expertise, certification reports have a high likelihood of containing incorrect or misleading information.

Minimal required documentation includes certification findings and security operating procedures. The latter are needed so that security managers know what organization-specific rules to follow in initializing and using security permissions and audit capabilities.

2.2.6 Support Accreditation

The last step of acquisition and integration is to support accreditation or reaccreditation of the system into which the new capabilities have been incorporated. Accreditation is the management decision to operate the system [6]. Accreditation is based on certification findings. For initial capabilities there might be limited functionality or security restrictions that must be endured until later versions are available.

2.3 Operate System With New Capabilities

Capabilities finally are used in the operational system. Transition planning to integrate the new capabilities into an existing system is complex and must address training, procedures, data, hardware, and software. New procedures and roles might be needed. Data might have to be partitioned (e.g., into databases operating at different security classification levels). Old and new configurations might be operational simultaneously, with new MLS capabilities implemented for only a subset of the users. Care is needed that the insertion of MLS capabilities not disrupt operation.

MLS capabilities initially being fielded in operational systems should be carefully evaluated during the initial period of operation to assess the security, performance, and impact of the capabilities. Such scrutiny is needed because unforseen difficulties can arise when users, administrators, and security officers begin using a new capability to support an operational application. Another reason for careful oversight of the initial operational period is that current MLS capabilities, due primarily to limitations in product completeness and maturity, do not have the assurance of mature commercial products or of capabilities developed in accordance with a detailed, step-by-step development process. Subsequent releases of the MLS capability should improve its assurance, along with its functionality and performance.

Feedback is needed from initial MLS capabilities that have been fielded in operational systems. This feedback could be provided in the form of operational MLS experience reports, prepared about one year after initial fielding (or as needed). While there normally is no official requirement for such reports, the Joint MLS Technology Insertion Program encourages their preparation. The purpose of such reports is to record the view of real users rather than technologists or program planners. Whereas people who plan for or develop a capability might be inclined to put the best face on their efforts, people who *use* a capability should be better able to provide an objective assessment.

References

[1] DISA, September 1991, *Target Architecture and Implementation Strategy for the Joint MLS Technology Insertion Program*, Arlington, VA.

[2] DOD, 29 February 1988, *Military Standard Defense System Software Development*, DOD-STD-2167A, Washington, DC.

[3] Boehm, B. W., August 1986, "A Spiral Model of Software Development and Enhancement," reprinted in *Software Engineering Notes*, Vol. 11, No. 4, Association for Computing Machinery (ACM).

[4] Abrams, M., D. Akers, K. Bitting, A. Lee, J. Lovelace, and B. McKenney, September 1990, *Overview of Security in the Acquisition Process*, MTR-90W00138, The MITRE Corporation, McLean, VA.

[5] Caddick, E. M., April 1990, *Security-Relevant Documents Developed During System Acquisitions*, MTR10869, The MITRE Corporation, Bedford, MA.

[6] National Computer Security Center (NCSC), 21 October 1988, *GLOSSARY of Computer Security Terms*, NCSC-TG-004,Version-1, Ft. Meade, MD.

[7] NIST, 27 September 1983, *Guideline for Computer Security Certification and Accreditation*, FIPS PUB 102, Gaithersburg, MD.

Architectural Implications of Covert Channels

Norman E. Proctor and Peter G. Neumann
Computer Science Lab, SRI International, Menlo Park CA 94025

Abstract This paper[1] presents an analysis of covert channels that challenges several popular assumptions and suggests fundamental changes in multilevel architectures. Many applications could benefit from a practical multilevel implementation but should not tolerate any compromise of multilevel security, not even through covert channels of low bandwidth. With the present state of the art, the applications either risk compromise or forgo the benefits of multilevel systems because multilevel systems without covert channels are grossly impractical. We believe that the presence of covert channels should no longer be taken for granted in multilevel systems.

Many covert channels are inherent in the strategies that multilevel systems use to allocate resources among their various levels. Alternative strategies would produce some sacrifice of efficiency but no inherent covert channels. Even these strategies are insufficient for general-purpose processor designs that are both practical and multilevel secure.

The implications for multilevel system architectures are far-reaching. Systems with multilevel processors seem to be inherently either impractical or unsecure. Research and development efforts directed toward developing multilevel processors for use in building multilevel systems should be redirected toward developing multilevel disk drives and multilevel network interface units for use with use only single-level processors in building multilevel distributed operating systems and multilevel distributed database management systems. We find that distributed systems are much easier to make both practical and secure than are nondistributed systems. The appropriate distributed architectures are, however, radically different from those of current prototype developments.

Keywords covert channels, distributed systems, multilevel security, system architecture.

Introduction

This introduction describes covert channels and their exploitation. The next section gives some background on covert channel research and relevant standards. After that, we identify the circumstances in which covert channels need not be avoided when designing a system for an installation. First, we consider various reasons why covert channels might be tolerable in a multilevel system. Then, since covert channels are found only in multilevel systems, we consider when alternatives to multilevel systems are appropriate for an installation. This seems to leave a large class of installations that would want multilevel systems free of covert channels.

We next turn our attention to various reasons why multilevel systems have covert channels and consider how the needs of applications can be met without producing covert channels. We consider in particular how a multilevel system can allocate resources among levels without covert channels that compromise security and without inefficiencies that leave the system impractical. We describe the problems with dynamic allocation and identify three alternative strategies for secure and practical resource allocation: static allocation, delayed allocation, and manual allocation.

We describe some practical approaches to multilevel allocation for various devices, including multilevel disk drives, and explain why allocating software resources among levels is so troublesome. Finally, we present the implications for multilevel system architectures and suggest new directions for research and development.

To the Reader Earlier versions of this paper were misinterpreted by some very knowledgeable readers, leading us to clarify the exposition. Nevertheless, we warn readers familiar with the problems of covert channels in multilevel systems that, because we are questioning some popular assumptions about covert channels, what you already know about covert channels may cause you to misunderstand our main points. Thus, please forgive our belaboring certain central issues and slighting other fascinating topics that seemed less central to the discussion.

Covert Channels Covert channels are flaws in the multilevel security of a system.[2] The channels are found only in multilevel systems. A malicious user can exploit a covert channel to receive data that is classified beyond the user's clearance. Although a covert channel is a communication channel, it is generally not intended to be one and may require some sophistication to exploit. It may take considerable processing to send one bit of data through the channel; error control coding is needed to signal reliably through a noisy covert channel. Exploitation may require the help of two Trojan horses. One runs at a high level and feeds high data into the channel, and the other Trojan horse runs at a lower level and reconstructs the high data for the malicious user from the signals received through the covert channel. The low Trojan horse is not needed if the high one can send a straightforward signal that can be directly interpreted. Also, as we explain later, malicious users can exploit some special kinds of covert channels directly without using any Trojan horses at all.

A covert channel is typically a side effect of the proper functioning of software in the trusted computing base (TCB) of a multilevel system. Trojan horses are untrusted programs that malicious users have written or otherwise introduced into the system. A Trojan horse introduced at a low level can usually execute at any higher levels.[3]

A malicious user with a high clearance does not need to use covert channels to compromise high data. The mandatory access controls would permit reading the high data directly. Ordinary reading is certainly an easier way to receive the data if the discretionary access controls permit ordinary reading. If not, it is easier for a Trojan horse to copy the data into another place where the discretionary controls do permit the malicious user to read it than to exploit a covert channel to transmit the data.

The levels that concern us here are not necessarily hierarchical confidentiality levels. They may instead be partially ordered combinations of hierarchical levels with sets of compartments. We assume that a level might have some compartments. This means that two different levels may be comparable or incomparable. If comparable, one level is higher and the other is lower. If incomparable, neither level is higher or lower. A higher level denotes greater intended secrecy or confidentiality.[4]

Noise in Covert Channels The bandwidth of a covert channel is the rate at which information or data passes through it. A noisy channel intentionally or accidentally corrupts the data signal with errors so that the information rate is slower than the data rate. A very noisy channel with an apparent bandwidth of one bit of data per second might actually leak only one millionth of a bit of usable information per second. Such a low bandwidth is beneath the notice of some. A malicious user who might have received the classified answer to a yes-or-no question almost immediately if the channel had no noise would expect to wait almost twelve days for the answer. Of course, the channel still compromises security even though extremely high noise makes for an extremely low effective bandwidth.

Noise in a covert channel may also make its information probabilistic. For example, consider a slower covert channel with a bandwidth of a thousandth of a data bit per second where each bit received has a seventy-five percent chance of being the same as what was sent and a twenty-five percent chance of being wrong. A malicious user exploiting the channel must receive the answer to a yes-or-no question many times before believing whichever answer was received more often. The expected wait for each answer is about seventeen minutes, but it takes around five hours for confidence in the answer to reach ninety-nine percent. Here again, compromise of security is postponed but not prevented.

Background

Various approaches exist for detecting and analyzing covert storage channels [2, 12] and for avoiding some of them [5]. For covert timing channels, additional approaches exist for detection, analysis, and avoidance [4, 14]. Some approaches attempt to address both types of covert channels [3]. The notions of restrictiveness and composability [8] seek to preserve the absence of covert channels under composition, assuming their absence in the underlying components.

At the end of this paper, we discuss some new directions for multilevel system designs that avoid all

[2]Similar flaws in other aspects of security are sometimes called covert channels, too, but a covert channel in this paper is always a communication channel in violation of the intended multilevel policy of the system.

[3]If a program could run only at the level where it was installed, it would be harder for a malicious user with a low clearance to introduce the high-level Trojan horse. It would also be inconvenient to install legitimate software.

[4]For simplicity, we assume that levels are for confidentiality although they could instead be for integrity or for both integrity and confidentiality. The levels for mandatory integrity are duals of confidentiality levels; covert channels can compromise mandatory integrity in a direct parallel to their compromise of mandatory confidentiality. For example, a Trojan horse running at a low integrity level might covertly contaminate high integrity data where overt contamination was prevented by multilevel integrity.

covert channels. The architectures themselves are not new, of course. Others have considered similar architectures for somewhat different reasons [11, 13].

Much of the research and development in covert channels for practical systems has been devoted to reducing bandwidths to what some consider to be slow rates. Sometimes delays are introduced to lower bandwidth, and sometimes noise is added to lower the usable bandwidth. These approaches merely ensure that malicious users exploiting the channels do not enjoy the same quick response times to their queries as legitimate users enjoy. The assumption may be that if it takes hours or days for an answer to a simple illicit question, malicious users will ignore the covert channel and prefer more traditional methods of compromise, such as blackmailing or bribing cleared users. Although we do recognize some situations where covert channels are tolerable, we believe the reason is rarely because of low bandwidths. For most installations, we believe that all covert channels should be completely avoided, not simply made small. A clever, malicious user can generally compromise classified information with even the narrowest covert channel.

Other research in covert channels for practical systems has addressed the elimination of some specific varieties of channels. The other varieties, typically including all timing channels, are permitted in a multilevel system because the developers could not find a way to eliminate them without rendering the system impractical for its legitimate functions. The assumption may be that any channel that is too hard for a developer to eliminate must also be too hard for a malicious user to exploit, but this assumption is so clearly fallacious that it is never explicit.

A cynical interpretation of this willingness to tolerate residual channels is that, because many users have simply accepted systems with covert channels despite the potential for security violations, developers treat a multilevel security policy as an ideal to approach, not as a requirement to meet. A more generous interpretation is that the developers intend to eliminate more and more kinds of covert channels with each new generation of multilevel designs hoping that someday they can actually design a system with no covert channels. We wish they would go straight for systems free of covert channels, and we believe the goal can be reached.

Standards The U.S. Defense Department standards in the *Trusted Computer System Evaluation Criteria* [9], also known as the Orange Book, place restrictions on covert channels in secure systems. Systems evaluated at classes C1 and C2 would have no covert channels simply because they would always be run at a single level. There are no restrictions on covert channels in a class B1 system, even though the system would probably have plenty of them.

For a class B2 system, an attempt must be made to identify the covert storage channels, measure their bandwidths, and identify events associated with exploitation of the channels. The design must avoid all storage channels with bandwidths over one bit per second, and the audit must be able to record the exploitation events for any storage channels with bandwidths over one tenth of a bit per second. There are no restrictions on covert timing channels. In a class B3 system, the criteria for covert channels are extended to the timing channels.

In a class A1 system, the attempt to identify covert channels must use formal methods, but the criteria are otherwise the same as for class B3. The requirement of formal methods does imply that the informal methods acceptable for classes B2 and B3 may miss some covert channels. Among the channels that formal methods themselves tend to miss are the timing channels.

The criteria for covert channels in other security standards are similar to the Orange Book criteria. Although no standards require avoiding all covert channels, considerable theoretical work has been done on hypothetical systems free of covert channels. This is in part because absolute multilevel security would be better than multilevel security with potential compromise through covert channels. Another reason is surely that absolute security is far easier to express in a mathematical model than is compromised security.

We feel that the tolerance of covert channels in security standards is unnecessary and therefore inappropriate for most multilevel systems. In fairness, when the Orange Book was written, covert channels were believed to be inevitable. This belief remains widespread today. We do not accept the inevitability of covert channels in practical multilevel systems, and we fear that the current tolerance of covert channels is itself a major threat to classified information. The Orange Book and other standards are meant to promote the development of secure systems. The standards should not be used as excuses for developing systems with unnecessary flaws.

Tolerating Covert Channels

A malicious user who is cleared for certain classified data can always compromise the secrecy of the data. The problem with covert channels is that a malicious user with the help of one or more Trojan

horse programs can exploit a covert channel to compromise data classified beyond the user's clearance. Installations without malicious users or without Trojan horses can tolerate whatever covert channels a multilevel system might have because the channels would not be exploited.

No Malicious Users Of course, at any installation with more than one user, one can never be certain that no users are malicious, but a system-high installation might reasonably ignore its covert channels as if there were none. Since running system-high requires that all users be cleared for every level, the security officers of the installation would not expect users to exploit covert channels. To compromise any data in the system, a malicious user does not need a covert channel. Covert channels are tolerable in system-high installations because they do not increase the system vulnerability.

No Trojan Horses The security officers of some installations will assume that they have no Trojan horses. They may be right only because conventional compromise remains easier than exploiting Trojan horses when malicious users have limited technical skills. Few malicious hackers have access to multilevel systems, and few multilevel systems are exposed to malicious hackers. But security officers cannot know whether their installations are among the unfortunate systems.

An installation cannot reasonably be assumed free of Trojan horses unless appropriately trained people rigorously check all the programs that run on the system to be sure that none harbor Trojan horses. All new applications and all changes to existing applications must be reviewed. Rigorous reviews are so expensive and time-consuming that the software on the system must be fairly stable. Also, the system must not have any compilers, command interpreters, or similar programs able to create code and bypass the review procedures. Because no Trojan horses are available to exploit them, most covert channels are tolerable in an installation that can afford to ensure that all software is trusted not to contain a Trojan horse. Such a multilevel installation, if any exists, is probably dedicated to one modest-size application program running on a bare processor.

Malicious users can exploit some special covert channels to compromise certain kinds of classified data without employing Trojan horses. Typically, the data might indicate how busy the system currently is at various levels. If the data were only nominally classified, its leakage would not be serious, but release of such data at lower levels could constitute a real compromise of some systems. These special covert channels are, of course, intolerable even when an installation is known to be free of Trojan horses.

Low Bandwidth It may also be the case that leaks through covert channels are tolerable at some installations, provided the leaks are slow enough. The Orange Book suggests that covert channels with bandwidths under one bit per second are "acceptable in most application environments." This acceptability may simply be a concession to the sorry state of the art where some covert channels are sure to be present in any multilevel system and where merely identifying all the covert channels is generally infeasible.

It is difficult to believe that many security officers worry about how quickly data is compromised instead of worrying about whether it is compromised. Surely most worry about both problems. Nevertheless, a sufficiently low bandwidth could reasonably make covert channels tolerable at installations with special situations. Where all classified data is tactical data with ephemeral classifications, slow covert channels are tolerable if data would no longer be classified by the time it had been released. If leaking the answer to one crucial yes-or-no question is enough to compromise the system, either the classification of that answer must last only a split second or all covert channels must have extremely low bandwidth.

Similarly, at installations where a price tag can be placed on all classified data, some covert channels are tolerable because no Trojan horses to exploit the channels would be cost-effective or because any alternative without covert channels would be too expensive. If covert channel bandwidths are important in performing the cost-benefit analysis, some covert channels may be tolerable because of their low bandwidths. Where data is classified to protect national security, assigning prices is foolish and perhaps illegal.

Lack of Alternatives Many installations tolerate covert channels simply because every multilevel system under consideration has some and because those in charge feel they need multilevel systems. Fortunately, these difficulties can be overcome. We believe that there can be multilevel systems without covert channels and that there are often suitable alternatives to multilevel systems. The accreditors of automated systems for multilevel applications should not have to tolerate covert channels.

Alternatives to Multilevel Systems

Not all applications have to run on multilevel systems. We mention first two unattractive options that

must sometimes be taken. One is not to implement the application at all, and the other is to implement it with manual procedures only. The remaining alternatives are all automated implementations. The potential benefits of automation include convenience, accuracy, speed, and lower costs. These benefits have permitted the implementation of many applications that were infeasible before the advent of computers.

When an application involves only one level of data or when all users are cleared for every level of data, the best alternatives are a single-level system or a system-high system, respectively. But the applications that interest us here have some data classified at levels beyond the clearances of some users of the automated system. A single-level or system-high system cannot accommodate these applications, but a multilevel system is not the only alternative left. Another possibility is a system with an independent subsystem per level (ISPL). ISPL systems tend to be inefficient, but at least they are intrinsically free of covert channels. We present the ISPL architecture mostly because it is useful later for comparisons with more attractive alternatives.

In an ISPL system, there is a separate subsystem for any level where the system as a whole could have some data. Data is stored on the subsystem for the level matching the classification of the data. Additional upgraded copies of the data might be stored on some other subsystems at higher levels. A user has access to a subsystem only if its level is dominated by the user's clearance.

The subsystems are electronically independent. Each subsystem has its own hardware, and the hardware for the subsystem at one level is not connected to any hardware for subsystems at other levels. The subsystems are not completely independent, however. They are parts of a whole system with multiple levels because users sometimes refer to data on a lower subsystem in order to modify data on a higher subsystem. Users might also manually reenter data from a low subsystem into a high one, or operators might transfer data storage media to higher subsystems.

Like single-level systems, ISPL systems are inherently free of covert channels. Multilevel security is compromised only when people fail to follow proper procedures. The automated parts of the system cannot themselves reveal data to a user not cleared for it. However, trying to overcome some of the limitations of an ISPL system may lead to complex procedures, and the complexity brings serious dangers that accidental compromise would become frequent and that malicious compromise would become easy to arrange.

Because the subsystems are independent of each other, none of the coordination among subsystems can be automated. This tends to diminish all the potential benefits of automation. Unless the required coordination among subsystems is minor, an ISPL system may well be too inconvenient, inaccurate, slow, or expensive for an application. An integrated multilevel system may then be the only practical option. Unfortunately, multilevel systems typically have many covert channels.

Some Reasons for Covert Channels

Our aim is to avoid all covert channels in multilevel systems. Present experience, however, is that any practical multilevel system contains many covert channels, despite the attempts of developers to eliminate them. It has been so difficult to avoid covert channels because several highly desirable functions of a multilevel system seem to produce covert channels as a side effect. Fortunately, the essential multilevel functions can be implemented without building covert channels into the system.

The differences in functional capabilities between ISPL systems and multilevel systems highlight the major sources of covert channels in multilevel systems. In an ISPL system, which cannot have covert channels, the absence of connections among the independent subsystems for each level prevents the system from doing all that a multilevel system can do. Among the services requiring some manual assistance in an ISPL system are reading consistent data from lower levels, downgrading overclassified data, writing up reliably, and maintaining consistency among the values of data items at different levels. A multilevel system needs no manual assistance with these services, but the implementation techniques generally introduce covert channels.

Reading Down An automated system might allow one process to change data that another process is currently reading. Then, the value the reading process receives could reflect neither the value before the change nor the value after the change, but some useless mixture of the two values. Such mixed results from reading are unacceptable in most applications. The usual technique to prevent the problem is for the reading process to lock the data before reading it. The lock is not granted if any other process is currently writing the data, but once the lock is granted, no other process is permitted to write the data until the reading process releases the lock.

In a multilevel system with support for reading down, this technique produces a covert channel. Lower-level processes can detect when a higher process reads down to lower data because the higher

process holds a lock that prevents the lower processes from writing the lower-level data. Data cannot be locked for reading down without producing a covert channel.

Different techniques free of covert channels can ensure that high processes do not read inconsistent data [1, 6, 7]. The most popular technique is for the high process to check whether any lower process may have written the data between the time when the high process started to read the data and the time when it stopped reading the data. If so, the read is potentially inconsistent, and the high process repeats the entire read again until it is sure that no lower process wrote the data while it was being read. For some applications, there is a serious risk with this technique that a high process that tries to read a lengthy and volatile data item may keep trying to read the item for a long time without ever succeeding. Other techniques may be appropriate for those applications.

Downgrading All downgrading is inherently an exploitation of a covert channel. When the downgrading is legitimate, one could say that the channel is not really "covert," but the intended downgrade of overclassified data is often accompanied by some incidental and unacknowledged downgrading of other data. A Trojan horse might exploit the channel by manipulating the other data. It may also be possible for a Trojan horse to hide other data in the overclassified data. Multilevel system designs cannot provide legitimate automated downgrading and still avoid all covert channels.

Writing Up When a user working at a low level upgrades low data to a higher level, the data is said to be written up.[5] To make the writing reliable, the low user might be notified whether sufficient resources at the higher level are currently available to support the writing up. This notification produces an exploitable covert channel. Suppressing the notification makes writing up unreliable; the user or program that wants to upgrade data never knows whether the writing up worked or not. Applications that need writing up typically need reliable writing up, not hit-or-miss writing up. Reliable writing up can be achieved without covert channels by reserving sufficient resources at higher levels to accommodate all potential requests to write up. This is not easy to implement, and reserving the high resources may constitute a serious loss of efficiency. A practical multilevel system apparently cannot provide reliable writing up without covert channels.

[5] If the user were working at the higher level, the upgrade is from reading down, not writing up.

Consistency Across Levels When an application requires consistent values in two data items, a change to one may force a change to the other to keep them consistent, or alternatively, a change to one may be forbidden until after the other is changed to a consistent value. This can be problematic in a multilevel system when the two data items are classified at different levels [1]. If the levels are comparable, one approach is secure and the other produces a covert channel. Which is which depends on whether the data item changed is at the lower or higher level. Neither approach is secure if the levels are incomparable due to differing compartment sets.

Fortunately, one result of a rational classification of data is that any criterion of consistency applies to data items that are all at the same level. A data item would never have to be consistent with data items at any other levels. A requirement for consistency with a higher item implies that a user cleared to read the lower item can infer something about the higher item, which must have a consistent value. The existence of the inference suggests either that the lower data should be classified at the higher level or that the higher data should be classified at the lower level. If data were classified rationally, users cleared just for lower data could not infer anything about higher data.

In practice, however, classification is not purely rational, and some applications really may need consistency across levels. This can be achieved without covert channels, provided that reliable writing up is properly implemented and the levels involved are all comparable. The likely cost is gross inefficiency from keeping the writing up reliable and some inconvenience because users must always change the lowest items first. Data consistency across levels, freedom from covert channels, and practicality seem to be incompatible in a multilevel system.

Resource Allocation among Levels

We turn next to another distinction between ISPL systems and multilevel systems, their different abilities to allocate resources among levels. In an ISPL system, the allocation for a level is the hardware in the subsystem for the level. In order to change the allocation for a level, some piece of equipment must be replaced, and reallocating resources from one level to another is likely to involve bringing down two subsystems for a while. In a multilevel system, reallocating resources is more convenient. Resources can often be allocated to whichever level can make the best use of them at the time. This can greatly increase the efficiency of the system. With a multilevel system instead of an ISPL system, the users can get more

service from the same hardware or equivalent service from less hardware.

Reading down, downgrading, writing up, and data consistency across levels, as we explained before, are not just functional distinctions between ISPL systems and multilevel systems, but also common reasons for covert channels in multilevel systems. Similarly, resource allocation is a common reason why multilevel systems have covert channels, as well as being a functional difference from ISPL systems.

Because a system often has many kinds of resources, resource allocation may be the reason for most of the covert channels in a multilevel system. Among the space resources to be allocated are physical memory, entries in operating system tables software, storage on disk, and bandwidth in a network connection. The allocable time resources include processor time (CPU time), service time from the operating system, disk access time, and access time to other multilevel devices such as terminals, printers, tape drives, and network interface units. Resource allocation is a primary function of operating systems, but multilevel networks, database management systems, and even applications have resources of their own to allocate among levels.

We consider four general strategies for resource allocation among levels: static allocation, dynamic allocation, delayed allocation, and manual allocation. Dynamic allocation is the most efficient but inherently produces covert channels. The other three strategies are free of covert channels but can be inefficient to the point of complete impracticality when used for the wrong resources. Static allocation is the simplest strategy and the least efficient. It is usually as inefficient as an ISPL system. Delayed allocation and manual allocation are more efficient, sometimes approaching the efficiency of dynamic allocation. Delayed allocation is better suited to some resources, manual allocation is better for other resources, and a combination of both may be better than either one in some cases. We use the allocation of processor time as the main example to illustrate the four strategies.

Static Allocation With static allocation, a fixed portion of a resource is allocated to each level that shares the resource. One level cannot borrow from another level even when the first level could use more than its share and the other level has idle capacity.

If processor time is statically allocated, the share of time allocated to a level is generally determined through the initial system configuration. The configuration might assign time slots to each level. The schedule would consist of a sequence of time slots that is repeated for as long as the processor runs. The share for a level is the length of its time slot in the sequence or, if the level has several slots, their combined length. Only processes at the proper level run during the time slot for a level. The level gives up the processor at the end of its time slot even if some processes at that level still want processing time. On the other hand, during the time slot for a level, the processor is left idle whenever every process at the level is waiting for I/O or whenever there are no current processes at the level. This means that the processor may be idle during the time slot for one level when there are processes at another level that could have been serviced.

Dynamic Allocation At the cost of producing a covert channel, dynamic allocation avoids such wasting of resources. Resources are allocated among levels based on the current needs at each level. The simplest algorithms allow one level to borrow freely as needed from other levels. More complicated dynamic allocation algorithms place some limits on how much can be shared or how frequently reallocation can occur.

If processor time is dynamically allocated, the current loads might freely determine the share of processor time for a level, or the system may adjust shares within configured limits. When the higher levels are busy, processes at lower levels cannot get as much processing time as when the higher levels are idle. Because lower processes can detect whether higher levels are relatively idle or relatively busy, there is an exploitable covert channel.

For example, a high Trojan horse could send a "one" bit during a particular period by requesting so much processor time that the processor would seem especially busy to the low Trojan horse receiving the signal. To send a "zero" bit instead, the high Trojan horse would refrain from requesting processor time so that the low Trojan horse would find the processor relatively idle. Irregular patterns of legitimate activity probably make the channel noisy, and the noise reduces the effective bandwidth of the channel. But the channel is not eliminated. Some bandwidth would still be available for leaking information to users who are not cleared to see it.

The covert channel from dynamic allocation is exploited by exhausting the resource. Processor time like any resource is finite, but in some cases, processor time is effectively inexhaustible. If the heaviest possible load on the processor would not consume all the available time, there is always time available whenever a level wants some. This eliminates the covert channel, but it makes dynamic and static allocation equally inefficient. Ensuring that processing time is always available with dynamic allocation

would ensure that time is always available with static allocation, too. The same amount of processing time would go idle either way.[6]

Delayed Allocation Allocating resources to one level may entail denying the same resources to other levels that request them later. A dynamic allocation strategy that could support instant reclamation of resources need not have a covert channel. Each level would have a basic allocation, but when a lower level was not using all of its basic allocation, a higher level wanting more than its own allocation could borrow from the unused portion of the lower level allocation. If the lower level later became busy enough to want some of the borrowed allocation back, enough would be instantly reclaimed for the lower level.

Similarly, if an intermediate level wanted more than its allocation, it could also borrow from the lower level. When a higher level had already borrowed from the lower level, that would not influence how much the intermediate level could borrow. If necessary, resources that were borrowed for the higher level would be instantly reclaimed and reallocated to the intermediate level.

A higher level could not borrow resources from a lower level while the lower level was using them or while any intermediate level was already borrowing them. Also, a lower level could never borrow from a higher level although it would sometimes reclaim its own basic allocation from the higher level or usurp the resources of a still lower level that the higher level happened to be borrowing.[7]

When a process at a level is given resources, it might be told whether they come from the basic allocation for its own level, and if not, it could be told from which lower level it is borrowing them. It must not be informed whether the resources were reclaimed from a higher level. There is no covert channel because the borrowings of higher levels do not affect the resource amounts available for a lower level.

When requests for resources are satisfied, the resources are allocated with the same speed whether the resources are currently free or currently being borrowed at a higher level. If free resources might be allocated instantaneously, then borrowed resources must be reallocable to a lower level instantaneously, too. Since instantaneous reallocation is not feasible for most resources, instantaneous allocation of free resources usually cannot be provided either. If borrowed resources can be reallocated only slowly, free resources must be allocated just as slowly. The delayed allocation strategy is named for the sometimes substantial delays the strategy can introduce in the allocation of resources.

For a delayed allocation of processor time in a system with only comparable levels, throughput could be maximized by making a basic allocation of all the processor time to the lowest level. Each level would seem to have available to it all the time that lower levels were not already using. At the end of each time slice, the processor would be allocated to the lowest level with a process ready to run.[8] An interrupt for the currently allocated level could be serviced promptly, but an interrupt for another level would not be serviced until the next time slice when no lower tasks were pending. With all time slices being of equal duration, this delay in servicing interrupts conceals whether the processor was idle when the interrupt occurred or was busy servicing a higher level. The delay clearly wastes some processor time in order to avoid the covert channel found with dynamic allocation.

Since a lower level would not be prevented from consuming all the time and shutting out all higher levels, some installations may prefer instead to give each level a basic allocation in order to guarantee some

[6] In some circumstances, dynamic allocation might always give enough time even though static allocation of the same total capacity did not always give enough. This may occur if the limits on the load yield a maximum combined load for all levels that is less than the sum of the maximum loads for individual levels. The most likely reason for such a pattern of loads is that some other dynamically allocated resources are exhausted. The allocation routines for the other resources would then have exploitable covert channels even though the allocation routine for processor time did not.

[7] When a system involves incomparable levels, the rules for borrowing are more complex. Incomparable levels cannot borrow from each other, nor can they compete to borrow from another level lower than them. One way to avoid competition among incomparable levels is to allow only some of the higher levels to borrow from a lower level. The system configuration would select which higher levels can borrow from a level. The levels selected to have borrowing privileges for a resource at a lower level must be mutually comparable. For any two incomparable levels, the selections for a lower resource might contain one or the other of the two incomparable levels, or perhaps neither, but certainly not both. Because any level not selected could not borrow the lower resource at all, it would never compete for the resource with another incomparable level that was selected.

[8] All levels except the lowest level are borrowing their time from the basic allocation to the lowest level. Since two incomparable levels cannot compete for the same resource, a system with incomparable levels needs some changes to the algorithm. The simplest variation is to specify a repeating sequence of time slices. The slices in the sequence need not all be the same length of time, but for each cycle through the time slices, each slice must be the same length as it was in the first cycle. All the time slices would still be in the basic allocation for the lowest level, but different sets of borrowing levels should be selected for different time slices in the sequence to ensure that each incomparable level has chances to borrow processor time.

time for each level. This fairness comes at the cost of lower overall efficiency. Whenever multiple levels compete for a shared resource, any strategy to prevent denial of service to high levels will either require more resources or produce a covert channel, entailing compromise of multilevel security.

The advantage of dynamic allocation is its more efficient use of processor time than with static allocation. In fortunate circumstances, delayed allocation is essentially as efficient as dynamic allocation. But in ordinary circumstances, the delays introduced to conceal processor loads at higher levels make delayed allocation less efficient than dynamic allocation. And in unfortunate circumstances, delayed allocation could be even less efficient than static allocation.

Manual Allocation A contributing factor in producing a covert channel with dynamic allocation is that the allocation is changed automatically based on data from untrusted software. Changes in the allocation based on trustworthy data do not necessarily produce a covert channel. The operators of a multilevel system could sometimes switch the system manually among a variety of different multilevel allocations appropriate for different situations. The operators would choose an allocation based on their expectations of the upcoming resource needs at each level. They must be careful to use information from outside the system, not simply the current loads at each level. Those loads may reflect the influence of Trojan horses instead of legitimate activity.

More automated variants of manual allocation are also possible. Some information within the system could be used for automatic changes in the allocations of resources among levels. The information that is safe to use is information that users or operators input manually and that comes through trusted paths to ensure freedom from the influence of any untrusted software. On a multilevel system, safe inputs may include user logins, user logouts, user requests to change to a new level, and possibly some other inputs through an operator console.

These inputs must follow trusted paths from the user or operator to the TCB. There is no covert channel because Trojan horses are incapable of spoofing what a user does through a trusted path. That is precisely what makes a path qualify as a trusted path. Since Trojan horses cannot produce any of the manual inputs that determine how allocations are updated in the manual allocation strategy, they cannot influence the changes in allocation to any level. It is crucial that the only information used to adjust the allocations is information the operating system receives directly from users through trusted paths.

Manual allocation of processor time can be reasonably efficient in a multilevel system used primarily for online processing. If the user inputs for logging in, logging out, and changing level all come via a trusted path, the allocation of processor time for a level can be proportional to the number of user sessions currently logged in at a particular level. This is often a fair measure of the expected load at that level. No time would go to levels with no current user sessions. When all current sessions are at one level, that level would be allocated all the processor time. Allocations would be subject to change each time a user logged in or out or changed from one level to another.

The ratio of the number of current user sessions at a level to the total number of current sessions is a secure basis for manual allocation only on a system where the total number of users logged in is unclassified. If users with low clearances must not know how many users are logged in at higher levels, then the ratio determining the allocation for a level should instead compare the current sessions at the level to the sessions at or below the level. Manual allocation based on this ratio would be somewhat less efficient.

Efficiency might be enhanced by taking into account some other information about current user sessions that the trusted paths have validated. The user's name, the time of day, and, if the system is distributed, the processor supporting the user session could be used to anticipate different loads from different sessions and calculate allocations based on those expectations. The weights for the calculations should come from tables the operators have prepared in advance, not from the current demands of the sessions.

In a multilevel system where online processing predominates but there is some background or batch processing, this approach should be modified so that some time is allocated to levels that may have offline processing. Otherwise, offline processing at a level would cease whenever there happened to be no current user sessions at the level.

Reallocation based solely on manual inputs would not be as efficient as dynamic allocation based on all available information. It should still be more efficient than a static allocation that never changes. Manual allocation, like delayed allocation, is less efficient than dynamic allocation. Both allocation strategies are compromises between dynamic allocation and static allocation.

Manual and delayed allocation can be combined. The same kinds of inputs as the manual strategy uses to update allocations can be used to update the basic allocations for the delayed strategy. The hybrid allocation strategy improves the efficiency of delayed

allocation, and with resources for which delayed allocation is appropriate, the hybrid strategy is more efficient than manual allocation, too. The hybrid strategy cannot outperform the best dynamic allocation algorithm, nor is it likely even to be equally efficient. However, the covert channels of dynamic allocation are absent from a combination of manual and delayed allocation, just as they are with static allocation, simple delayed allocation, and simple manual allocation.

Allocating Device Resources

We call a device multilevel if it ever stores or transmits data for more than one level. At one extreme, the device may always handle hundreds of levels, or at the other extreme, it may handle one level on some days and another level on the other days.

As a first example of a multilevel device, we consider a multilevel terminal. It is inconvenient for a user to move to a different terminal in order to work at a different level or for the user to have as many terminals on one desk as there are levels of work to do. With one multilevel terminal, terminal access time could be allocated to whichever level the user currently wants. Multilevel terminals would cost more than single-level terminals, but the convenience may justify the added cost. And if one multilevel terminal fully replaces several other terminals, there may even be a cost savings.

The multilevel terminal would need some special manual inputs for selecting the level where the user wants to allocate the terminal access time. A reset button, a dial or switch for indicating a level, and a ready button would be enough. When the user presses the reset button, the terminal clears its screen and any volatile memory, locks the keyboard, and unlocks the level dial. Then, the user can set the dial to the new level. When the user presses the ready button, the terminal locks the dial, selects the single-level communication line at the level corresponding to the setting of the dial, and unlocks the keyboard.

When the terminal is installed, the security administrators should make sure that the dial settings correctly label the processors that can be accessed through the corresponding single-level lines. The terminal must also be protected from sabotage, of course. We caution against making the multilevel terminal too sophisticated. A multilevel workstation is far less likely to be implemented free of covert channels than is a basic multilevel terminal. Pushing the reset button must remove all traces of whatever had been done before.

A similar approach would work for a multilevel printer or multilevel tape drive. The reset button of a printer must clear all physical traces of what was printed at the previous level. The justification for a multilevel printer or tape drive is probably lower cost or greater convenience again.

Trusted Network Interfaces A network of multilevel lines is more convenient for operators to install and maintain than are separate networks of single-level lines for a variety of levels. The convenience may justify the cost of the trusted network interface (TNI) units to connect each single-level communication line to a multilevel line. Especially in a wide-area network, the savings from having fewer cables may also offset the cost of TNI units.

If a multilevel line is a radio-frequency cable, each level can be statically allocated its own frequency band. A TNI unit would tune to a band based on its control settings. Whoever installs or maintains a unit connecting a multilevel line to a single-level line must check that the control settings of the unit agree with the level of the single-level line.

TNI units should be connected to the communication lines of single-level processors and devices so that they can communicate over the multilevel network lines. Rather than having TNI units connected to the various single-level lines for a multilevel device such as the terminal described earlier, one TNI unit could be embedded in the multilevel device so that one multilevel line could replace all its single-level lines. The terminal would retune its frequency based on the current dial setting when the user pushed the ready button. Embedding a TNI unit is also an option for a multilevel printer or multilevel tape drive.

A network of multilevel lines with TNI units wherever processors and devices connect to the network is functionally equivalent to separate single-level networks. A single-level processor could communicate with other single-level processors and devices only if they are at the same level. It could communicate with the multilevel devices we described only when they were currently allocated to the same level, too.

More complex TNI units might support multiple single-level lines or support an allocation strategy for the multilevel lines more efficient than static allocation of frequency bands to levels. We suspect the added efficiency would not offset the problems of the extra complexity: a higher cost per unit and reduced assurance of multilevel security.

Cryptographic methods can supplement such TNI units but are never a substitute. If network lines are vulnerable, encryption can help preserve the confidentiality and integrity of messages transmitted over the network. However, if the network does not carefully allocate resources based on the levels of the decrypted

messages, there are covert channels. Users communicating at low levels could detect heavier and lighter loads on the network from activity at higher levels, possibly due to Trojan horses. Encrypting messages does nothing to eliminate this covert channel.

Multilevel Disk Drives Any multilevel application requires some support for reading down. Reading down can be implemented with multilevel processors, multilevel disk drives, some other multilevel storage media, or a combination. Disks are more generally useful for reading down than are other storage devices. Also, we believe that multilevel disk drives are much easier to build free of covert channels than are multilevel processors. We are not certain that multilevel drives really can be implemented without covert channels as nobody has yet tried, but we sketch a design that seems feasible.

The design uses manual allocation of the storage space on the disk and uses a combination of delayed and manual allocation for the access time to the disk drive. The interface for the operator has a reset button, a restore button, an accept button, and various browsing buttons to control a display panel. The interface to the rest of the multilevel system is through separate single-level lines for each level the drive supports.[9]

A special single-level line connects the disk drive to a single-level processor with a configuration table that the operator maintains. The table shows (1) the levels of the other single-level lines, (2) which levels are higher or lower than other levels,[10] (3) what level of data is to be stored in each sector of the disk, (4) how long each period in the access time schedule lasts, (5) which level is the basic level for each time period in the schedule, (6) which higher levels may borrow time during each time period,[11] and (7) what position the disk arm is to be in at the end of each time period.

When a configuration table takes effect, the allocation of storage space to a level is the sectors that the configuration assigns to that level. The allocation strategy for access time is a hybrid of delayed and manual allocation. The effective configuration gives the parameters for delayed allocation. The basic allocation of access time to a level is the time periods where that level is the basic level.

[9]As before, the single-level lines could be replaced with an embedded TNI unit and a multilevel line.

[10]The level of the special line should be lower than the levels of the other lines.

[11]If the disk supports some incomparable levels, the borrowing levels for a time period must be chosen to be mutually comparable.

While the disk drive is providing its regular reading and writing services, the drive rejects any requests to change its internal configuration table. When the operator pushes the reset button, the disk drive locks all the buttons, stops regular reading and writing services, and waits to receive a new configuration table through its special line. The operator working on the processor where configuration tables are maintained should request a change to the new configuration. If the disk drive finds the new configuration unacceptable, it shows an error code in its display panel and unlocks the reset and restore buttons. The operator has a choice of fixing and resubmitting the new configuration or restoring the old configuration.

If the drive would accept the new configuration, it unlocks all buttons and prompts the operator to double check the changes. The operator uses the browsing buttons to check all parts of the new configuration and perhaps also the old configuration to be sure that the configuration the disk drive received is exactly as intended. This precaution means that the single-level processor where the table is maintained and the path connecting the processor and disk drive do not have to be completely trusted.

If the configuration does not look right, the operator pushes the restore button. The disk drive locks the restore and accept buttons, discards the new configuration, and resumes regular service with the old configuration. If the operator pushes the accept button instead, the restore and accept buttons are still locked, but it is the old configuration that is discarded and the new configuration that is used to resume regular services. Also, before resuming regular reading and writing services with a new configuration, the drive clears any disk sectors then allocated to levels lower than before.[12] During regular services, the reset and browsing buttons remain unlocked.

While the disk drive serves a level, it accepts inputs and returns outputs through the communication line for the level. The other communication lines are ignored. The drive honors any requests to read or write sectors at the current level. To support reading down, the drive also honors requests to read sectors at lower levels.

Within the disk drive itself, there is a scheduler that determines which level to serve and for how long. The scheduler cycles through the schedule of time periods in the current configuration. At the beginning of

[12]Any sector allocated to a level incomparable to its old level is also cleared. If the level of a sector is left unchanged, its contents are kept. The contents are also kept in a sector whose level increases. In such a sector, the contents are effectively upgraded to the higher level.

a time period, it serves the basic level for the period. When appropriate, the scheduler may change level before the period ends and allocate whatever remains of the period to the lowest level that can borrow time in the period. It may also change level more times and allocate the remainder of the period to the next highest borrowing level.[13] If the highest borrowing level for the period is reached, the level stays the same until the start of the next period – when it becomes the basic level for that period.

The scheduler in the disk drive changes to the next highest borrowing level when the current level has no more disk accesses to make. If the current level is already the highest borrowing level, the drive waits idly until the period ends or more requests are received at the highest level. The drive does not change level if there would not be enough time to establish the new higher level and still position the disk arm as the configuration requires before the period ends. Similarly, as the period draws to its end, the disk drive rejects any access request that could not be completed in time to position the disk arm properly afterward.

The covert channel that would be produced by a dynamic allocation of access time is not found in this design. The allocations of storage space on the disk and the parameters used for delayed allocation of access time change only when the configuration changes, and that is only when the operator pushes the appropriate buttons. While the configuration remains unchanged, the performance of a disk drive in one time period has no effect on its performance in later time periods. Within a time period, the service to a level depends just on the requests from that level and lower levels. The higher borrowing levels receive no service until the lower levels voluntarily release their claims on the time period.

The sometimes long delays while a multilevel disk drive is inaccessible from a level make the drive inappropriate for the I/O of many ordinary processes. We suggest that most data be kept on single-level disks and accessed there primarily. Multilevel disks would hold only replicas of data that is sometimes read down. The following scenario explains how this might work.

A Scenario with Upgraded Replicas An ordinary process running on a single-level processor at some low level writes to a file stored on a single-level disk at the low level. When the process releases its write lock, a new value of the file is available for other processes at the low level to read from the same disk. But if the file header indicates the file is replicated, the replicas do not yet have the new value.

A replica management (RM) process on the same processor sends the updates to RM processes for any other disks that the file header indicates keep replicas at the low level. Although some of these RM processes may run on other processors, all run on single-level processors at the low level. The RM processes update the replicas on their disks to reflect the new value of the file. Multiple copies at the low level increases the availability of the file to users throughout the system. If its disk is multilevel, an RM process also records the new time stamp of the updated replica in a special disk segment for the low level.

Periodically, each process of another kind, the upgraded replica management (URM) processes, reads down on a multilevel disk in the time stamp segments for any levels lower than the level of the processor where the URM process runs. For each file with an upgraded replica at the high level of its processor, the URM process checks whether the time stamp of the lower replica has changed since last checked. If so, the URM process reads the updated lower replica of the file. It is again reading down on the multilevel disk.

The URM process sends the updates to the appropriate RM processes at the high level. As before, the RM processes write the new value of the file into the replicas on their disks at the high level. If any of these disks are multilevel, that may trigger another round of propagating the updates to replicas at still higher levels.

The new value of the file becomes available to ordinary processes running on single-level processors at a variety of levels. A process running on a processor at one of those levels can read any replica of the file found on a single-level disk at the same level.[14]

In the scenario above, all processes can run on single-level processors. Ordinary processes can do all their reading and writing on single-level disks. The only processes that must access multilevel disks are the replica management (RM) and updated replica management (URM) processes. An RM process reads and writes time stamp segments and replicas at its own level, and the URM processes read down to lower time stamp segments and lower replicas.[15]

[13] Because levels that may borrow time within a period are chosen to be mutually comparable even when the drive supports incomparable levels, the next highest borrowing level is uniquely defined until the highest borrowing level is reached.

[14] If the single-level disk is remote from the process, processes on other processors at the same level would help with the reading.

[15] A disk controller process on the same processor as the RM or URM process might mediate its reading and writing of the multilevel disk.

The inefficiencies of the allocation strategy for access time to the multilevel disk drives may hinder the upgrading of new or changed files. To update the upgraded replicas at the same time as the changes are made in the file itself would require reliable writing up, not just reading down. Because a covert-channel-free system is not expected to have reliable writing up, there will be some lag between the writing of a file and the updating of the upgraded replicas. The choice of an allocation strategy for the multilevel disk drives would affect only how long that lag can be. It does not affect any other processing. In particular, the I/O of ordinary processes and the propagation of replicas within a level are unaffected. They can benefit from all the efficiencies of high-performance, single-level disks.

Allocating Software Resources

While discussing multilevel devices, we have ignored multilevel processors and assumed that the multilevel devices would have to communicate with single-level processors. We now consider some of the resources of a multilevel processor. A multilevel processor has a trusted computing base (TCB), typically consisting of a kernel and some trusted processes. The software for the kernel and most trusted processes runs multilevel. The resources of that software are allocated among the various levels that the software serves.

As with hardware resources, dynamically allocating these resources on the basis of current demand creates an exploitable covert channel. Since the resources are limited, a low process employing the services of the multilevel software can detect how much has been allocated to higher levels, and a high process can send signals by modulating its demands on the multilevel software services. Static, delayed, or manual allocation, on the other hand, would produce no covert channels. Static allocation is feasible for most TCB software resources but is relatively inefficient. Manual allocation is often feasible and more efficient. Delayed allocation is also more efficient but would be too difficult to implement correctly for many software resources.

Kernel Resources The innermost layers of a trusted operating system for a multilevel processor are called a trusted executive or kernel. The layers that concern us include the layer presenting the abstraction of processes and all lower layers. These are the layers that do not run as processes. The kernel is inherently multilevel, and many of its resources are also multilevel. The execution time of the kernel is allocated among the levels. An allocation of processor time to a level includes the time the kernel spends serving that level, not just the execution time of single-level processes at the level. The storage resources of the multilevel kernel in a multilevel processor include most of the system data space. At any given moment, some of these resources would be fully allocated to the same level as is the processor time. Other storage resources might be partially allocated among levels.

It is extremely difficult to avoid every covert channel in the allocation of kernel time and storage in a multilevel processor. Some kernel resources can easily be allocated among levels using a static or manual allocation strategy, but it is unlikely that all resources of a practical multilevel kernel would be so safely allocated, especially in the lowest layers of the kernel.

A multilevel processor embedded in a special-purpose device such as a disk drive, printer, terminal, or network interface unit should need such a simple executive that safe allocation of all resources can be achieved without sacrificing practicality. The executive probably would not even support real processes.

A more general-purpose multilevel processor supporting user processes, however, seems doomed to have some covert channels at least within its kernel. The service time and data spaces for the lowest kernel layers could not avoid load-influenced dynamic allocation. The covert channels might all have small bandwidths or high noise, but they would still be there for malicious users to exploit, however slowly. Even some special-purpose multilevel processors, such as file servers, may be too sophisticated to be reliably free of covert channels.

To date, no designers have even come close to producing a covert-channel-free kernel for a multilevel operating system. In a typical design for a multilevel kernel, many low-bandwidth covert channels are not even identified.

Trusted Process Resources Secure allocation among levels is somewhat easier for the resources of multilevel trusted processes than for kernel resources. This may be largely irrelevant, however, because multilevel processes exist only on multilevel processors with more sophisticated kernels. Since the kernels already would have introduced some covert channels, the effort to avoid all covert channels in the trusted processes may be futile. The result would still be a TCB with some covert channels.

As with the kernel, the allocation of the execution time of a trusted process to a level must be considered part of the allocation of processor time to the level.

Static allocation of trusted process time is simpler, but the efficiencies of manual allocation might justify the extra complexity.

The virtual address space of a trusted process in a multilevel processor gives it storage resources that can be allocated among the levels that the process serves. Some variables in the address space would be fully allocated at any moment to the same level as the process time. Other storage resources, especially structures such as tables, lists, and buffers, might be partially allocated among levels based on a static allocation, or perhaps a manual allocation. Dynamic allocation based on current demand would create a covert channel, of course.

Memory management for the address spaces of trusted processes differs from the memory management for single-level process address spaces. Because the storage resources of a trusted multilevel process are allocated among multiple levels, it is not safe to handle them like those of untrusted single-level processes. The level of an untrusted process labels its whole address space, but the labeling of trusted process storage is not so simple.

The data of a trusted process must always be clearly labeled when it is stored in physical memory, when it is communicated over the memory bus, when it is kept on a paging disk, or when it is sent over communication lines between the processor and the paging disk. Otherwise, it becomes impossible to maintain control over the allocations among levels for various resources, including space in physical memory, access time to the memory bus, storage space on the paging disk, access time to the paging disk, and access time to the lines connecting the processor and the disk. Without explicit labels on trusted process data at all times, current demands would influence the allocation of those resources. Their allocation strategies would degenerate into some variety of dynamic allocation with covert channels and compromise of multilevel security.

Architectural Implications

Avoiding all covert channels in multilevel processors would require static, delayed, or manual allocation of all the following resources: processor time, space in physical memory, service time from the memory bus, kernel service time, service time from all multilevel processes, and all storage within the address spaces of the kernel and the multilevel processes. We doubt that this can be achieved in a practical, general-purpose processor. Perhaps the simplest strategy, static allocation, would be possible, but then the multilevel processor is not significantly more efficient than a set of single-level processors. It would be better to replace it with single-level processors and have real assurance of freedom from covert channels in processors. We suggest that multilevel systems not have any multilevel processors.

Having no multilevel processors certainly helps to minimize the TCB for mandatory security. This is especially appropriate for the high-assurance systems at the Orange Book classes B3 and A1. Because of the rapid drop in prices for processors and memories and the relatively wide selection of secure single-level processors, limiting a multilevel system to single-level processors may impose little or no penalty in efficiency. We believe the best architecture for most multilevel applications is a Distributed, Single-level-processor, Multilevel-secure (DSM) system. Even if a multilevel application does not need a distributed architecture for any other reason, we feel it should be distributed in order to be multilevel secure.

The network in a DSM system must not introduce covert channels. A simple option is a separate network for each level to connect the single-level processors at that level. A potentially less costly network has multilevel lines connecting all the processors and has the trusted network interface (TNI) units sketched earlier ensuring covert-channel-free allocation of the lines. The two options are functionally equivalent. The difference is in the number and capacity of the lines and in the hardware at the interface between the processors and the network.

Multilevel System Benefits in DSM Systems

Each processor of a DSM systems handles just one level, as in an ISPL system. An important question is whether a DSM system is as limited in its functionality as an ISPL system.

Downgrading, writing up reliably, and maintaining data consistency across levels cannot be fully automated as they can be in systems with multilevel processors and covert channels, but they can at least be more automated than in an ISPL system. Many, perhaps most, multilevel applications require none of these functions, but some do need one or more of them. Manual contributions to reliable writing up or to data consistency are inconvenient, but the only practical alternatives compromise multilevel security. Downgrading is so fraught with risk that it is reasonable to insist that some critical step be performed manually. The inconvenience is worthwhile.

Reading down is the essence of multilevel processing. Users perceive a system as multilevel if they have a choice of levels at which to work and if they can refer to the data at lower levels while creating or updating data at the current working level. Reading down and

ordinary single-level services are sufficient for most multilevel applications. DSM systems need not have the same problems with reading down as ISPL systems do. Reading down can be supported with multilevel disk drives similar to those described earlier. However, most disk drives in a practical DSM system should probably still each service a single level.

Some multilevel hardware in DSM systems can also escape the limitations on resource allocation in ISPL systems. Cost and convenience arguments justify static allocation of multilevel network lines and manual allocation of such resources as terminals, tape drives, and printers.

Partitioning Levels In the classification scheme of the U.S. Department of Defense, there are four hierarchical levels: unclassified, confidential, secret, and top secret. A level at which data is classified might also be one of the four hierarchical levels plus a set of nonhierarchical compartments. Many other classification schemes are similar. A user's clearance is the highest level of data the user may see. The clearance is the hierarchical level to which the user is cleared plus any compartments for which the user is cleared.

As noted above, it is best to run a multilevel application as system high if every user has the same clearance, covering all data levels in the application, no matter how many. However, a DSM system is appropriate when some users have different clearances and data is classified over a range of levels. Normally, a DSM system has different processors for each different data level. This is practical for many multilevel applications, ones with data at only two levels or at only a few levels. Some other applications, though, involving various nonhierarchical compartments use dozens or even hundreds of data levels. Processor prices may be falling, but a DSM system with at least one single-level processor for each of hundreds of levels would be impractical. However, a DSM solution may still be reasonable, provided that the number of different user clearances is fairly small, even though the number of different data levels is large.

We describe a DSM system with many data levels, many users, and a handful of different user clearances. A few users, perhaps just the system administrators, might be cleared for all levels, but most would have limited clearances. Probably, those clearances differ in their sets of compartments. The data levels are partitioned based on the overlaps and differences between pairs of clearances. Each partition contains one or more data levels; each data level belongs in one partition; and each clearance includes one or more complete partitions. In the best case, there are exactly as many partitions as clearances, but usually there would be more partitions.[16]

The processors are allocated, not to a single level, but rather to a single partition. A processor may handle data at every level within its partition and may communicate with any other processors sharing the same partition. It should have functionality similar to that required for class B1 in the Orange Book.

A user of a single-partition processor could be anybody whose clearance includes the partition. Because of how the levels are partitioned, the user's clearance will include all or none of the levels in the partition. This is why multilevel security is not compromised even though we expect the processor to have plenty of covert channels. The channels are tolerable because their exploitation could leak information only between levels in the same partition. A malicious user cleared for one level in a partition would not bother to exploit a covert channel in order to access another level in the partition because the user's clearance must include the other level, too.

Because covert channels can still leak within a partition, printed output from a partitioned DSM system can safely be released without review only if the label that the system generated is the highest level of the partition. Users can release output with other labels after manually confirming the labels.

Conclusions

Until feasible techniques are found to develop a covert-channel-free TCB for a practical multilevel processor, most multilevel systems should be DSM systems with some multilevel disks and perhaps other multilevel devices, but with no general-purpose, multilevel processors. The current research and development efforts on multilevel systems seem to focus on operating systems for multilevel processors, database management systems for multilevel processors, multilevel networks among multilevel processors, and distributed operating systems with multilevel processors. These systems are suitable only for installations that really must tolerate compromises of multilevel security through covert channels.

Promising directions for new efforts to serve secure installations include the development of multilevel disk drives and trusted network interfaces without covert channels. Other efforts should examine how single-level processors can use the multilevel disks

[16] In the worst case, n mutually incomparable clearances form $2^n - 1$ partitions. Probably, the levels in most of those partitions would never be used to classify any data in the system and so would never need resources. Partitions with no resource needs can be ignored.

and networks to build basic DSM systems that provide reading down in addition to the regular services of single-level distributed systems. Further efforts should enhance the basic DSM systems to build more sophisticated DSM systems or multilevel database management systems.

Because these implications for multilevel system architectures represent such a radical shift from the predominant direction of research and development, we encourage readers to dispute our conclusions. Optimists may wish to explain why most installations should tolerate covert channels or how a practical, general-purpose, multilevel processor can be developed with no covert channels. Pessimists may wish to explain why multilevel disk drives or trusted network interfaces cannot be developed without covert channels or why they could not be used to build practical DSM systems. We feel that avoiding all covert channels makes good sense for multilevel systems, that the current dismal state of the art is sufficient evidence of the unsuitability of architectures with multilevel processors, and that it is worth a serious effort to build a prototype of a covert-channel-free, multilevel system that has multilevel disk drives and single-level processors instead of multilevel processors.

References

[1] A. Downing, I. Greenberg, and T. Lunt. Issues in distributed system security. In *Proc. 5th Aerospace Computer Security Conference*, December 1989.

[2] R.J. Feiertag. A technique for proving specifications are multilevel secure. Technical Report CSL-109, Computer Science Laboratory, SRI International, Menlo Park, CA, January 1980.

[3] J.W. Gray III. Toward a mathematical foundation for information flow security. In *Proc. 1991 Symposium on Research in Security and Privacy*, pages 21–34, Oakland, CA, May 1991. IEEE Computer Society.

[4] W.-M. Hu. Reducing timing channels with fuzzy time. In *Proc. 1991 Symposium on Research in Security and Privacy*, pages 8–20, Oakland, CA, May 1991. IEEE Computer Society.

[5] P.A. Karger and J.C. Wray. Storage channels in disk arm optimization. In *Proc. 1991 Symposium on Research in Security and Privacy*, pages 52–61, Oakland, CA, May 1991. IEEE Computer Society.

[6] T.F. Keefe and W.T. Tsai. Multiversion concurrency control for multilevel secure database systems. In *Proc. 1990 Symposium on Research in Security and Privacy*, pages 369–383, Oakland, CA, May 1990. IEEE Computer Society.

[7] W.T. Maimone and I.B. Greenberg. Single-level multiversion schedulers for multilevel secure database systems. In *Proc. 6th Annual Computer Security Applications Conference*, December 1990.

[8] D. McCullough. A hookup theorem for multilevel security. *IEEE Trans. Software Engineering*, 16(6), June 1990.

[9] NCSC. *Department of Defense Trusted Computer System Evaluation Criteria (TCSEC)*. National Computer Security Center, December 1985. DOD-5200.28-STD, Orange Book.

[10] P.G. Neumann, N.E. Proctor, and T.F. Lunt. Preventing security misuse in distributed systems. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, March 1992. Project 1021, Final Report.

[11] R. Pike, D. Resotto, K. Thompson, H. Trickey, T. Duff, and G. Holzmann. Plan 9: The early papers. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, July 1991. Computing Science Technical Report No. 158. This report contains 7 conference papers presented during 1990 and 1991.

[12] P.A. Porras and R.A. Kemmerer. Analyzing covert storage channels. In *Proc. 1991 Symposium on Research in Security and Privacy*, pages 36–51, Oakland, CA, May 1991. IEEE Computer Society.

[13] J.M. Rushby and B. Randell. A distributed secure system. *IEEE Computer*, 16(7):55–67, July 1983.

[14] J.C. Wray. An analysis of covert timing channels. In *Proc. 1991 Symposium on Research in Security and Privacy*, pages 2–7, Oakland, CA, May 1991. IEEE Computer Society.

Assessing Modularity in Trusted Computing Bases[1]

J. L. Arnold, D. B. Baker, F. Belvin,
R. J. Bottomly, S. Chokhani, and D. D. Downs[2]

Abstract

In 1989, the National Security Agency (NSA) established a System Architecture Working Group (SAWG) to define and clarify the modularity criterion contained within the System Architecture requirement for Class B2 of the Department of Defense *Trusted Computer System Evaluation Criteria.* This paper summarizes the findings of the SAWG, which recommended that the following attributes be subjected to detailed analysis in order to assess modularity: code cohesion, complexity, coupling, data cohesion, duplicate code and data, and extraneous code and data.

1 Introduction

In 1989, the National Security Agency (NSA) formed a System Architecture Working Group (SAWG), whose primary mission was to review and clarify the modularity criterion of the System Architecture requirement specified in the *Trusted Computer System Evaluation Criteria* (TCSEC) [1] for classes B2 and B3/A1; specifically:

> **The TCB shall be internally structured into well-defined largely independent modules.**

The goal was to develop a definition of modularity for class B2 and any further definitions and clarification required for class B3/A1. The product was to be a report providing guidance to NSA teams tasked to examine and evaluate the modularity of systems designed to meet the TCSEC System Architecture requirement for classes B2 and above. This paper is a summary of the technical content of that report. [8]

The motivation for the TCSEC's modularity requirement is to achieve *understandability, maintainability,* and *testability*, rather than to provide some security functionality.

[1]This paper reports work conducted under funding from the National Security Agency.

[2]J. L. Arnold and R. J. Bottomly are with the National Security Agency, Ft. George G. Meade, MD; D. B. Baker (team leader) and D. D. Downs are with The Aerospace Corporation, Los Angeles, CA; and S. Chokhani and F. Belvin are with the MITRE Corporation, McLean, VA, and Bedford, MA.

Modularity adds assurance that the existent security functionality is understood and will remain intact through the lifetime of the system.

The most efficient and effective way to meet the modularity requirement is to first *design* the system using some form of functional decomposition. In fact, software engineering literature suggests that attempting to develop modular code without using a structured approach in its design will result in code that is *less* understandable, maintainable, and testable than code that constitutes a system built using a structured design discipline [10].

A "module" is defined simply as *one or more source code files*, and a "function" is a *callable entity, which may or may not return a value.* Although the Trusted Computing Base (TCB) comprises all the hardware, software, and firmware responsible for enforcing the system's security policy, the modularity requirement is not generally applied to hardware. Applicability of the requirement to firmware is determined on a case-by-case basis. Some of the factors considered are the amount of firmware in the system, the reputed reliability of the firmware (i.e., whether it is widely known to be reliable), the type of microcode (i.e., horizontal[3] or vertical), and the nature of the security functions the firmware implements.

In order to evaluate the modularity of TCB software, a collection of architectural evidence is examined, including the software engineering process purportedly used by the designers and developers, the system design documentation, the coding standards, and the contracts of the individual software modules. The SAWG defined "contract" as:

> A description of the overall purpose of a module. It includes the relationship between the input and output variables for all functions within that module and, therefore, describes *all* of the effects of the function. The input and output variables include not only the formal parameters of the functions, but also all state-maintaining variables, be they global to the system as a whole, or local to the function or module.

Evidence includes not only whether the documentation exists for each of these items, but also whether the documented disciplines are implemented and enforced.

The SAWG identified six attributes that play major roles in achieving a modular system. These attributes are shown in Table 1, which identifies for each attribute the

[3]Horizontal microprogramming is a technique whereby actions are encoded for multiple resources in a microinstruction. Horizontal microprogramming executes faster than vertical, but requires more complex decoding hardware. More importantly in this context, horizontal microcode is more difficult to code and to analyze than vertical microcode, which is similar to conventional programming [2]. To date, with respect to the architecture study, horizontal microcode has been considered part of the hardware.

level of abstraction at which it is examined and the acceptance criterion applied to it. When evaluating a *module*-level attribute, acceptability is based upon the strength of the attribute within individual modules (for code cohesion, complexity, and data cohesion) or in the interfaces between them (for coupling). *Function*-level attributes are evaluated for each function. *Code*-level attributes are evaluated relative to the source code as an aggregation of statements.

Modularity assessment is a continuing process encompassing the entire design and development process. The words of the modularity requirement are identical for classes B2 and B3/A1; therefore, relative to the attributes examined here, evaluation teams should expect to see no specific differences among these classes. However, B3/A1 system architectures are further constrained by the added requirements for TCB minimization, layering, abstraction, and data hiding. For B3/A1 systems, deviations are acceptable only if they do not adversely affect the system's ability to meet the modularity requirements and the additional system architecture requirements. Experience has shown that exceptions are fewer for the higher classes than for B2, due to the imposition of the additional constraints on the system architecture.

2 Evidence

In evaluating a system for modularity, a collection of evidence is examined and analyzed to evaluate the attributes identified in Table 1, and to ensure that the system was designed in a disciplined fashion and that the code was developed according to sound coding standards. Four types of evidence provide assurance that sound software engineering practices are in place:

- Coding Standards
- Contracts
- Design Documentation
- Software Engineering Discipline

For these types of evidence:

- The documentation must be clearly written (e.g., the design documentation should enable a reader to understand the design without having to look at the code).
- The documentation must be complete (e.g., the software engineering process should describe all processes involved in engineering the software from functional decomposition through final testing).

Attribute	Module	Function	Code	Criterion
Code Cohesion	x	x		- Functional, sequential, communicational cohesion acceptable - Temporal cohesion acceptable for specified cases - Logical cohesion acceptable only at module level - Coincidental cohesion unacceptable
Complexity	x			- Acceptable if module size and comprehension time within specified limits
Coupling	x			- Call coupling acceptable - Common coupling as determined by analysis - Content coupling unacceptable
Data Cohesion	x	x[a]		- Must exhibit at least logical data cohesion
Duplicate Code and Data			x	- No duplicate code or data
Extraneous Code and Data			x	- No extraneous code or data

[a]Note that Data Cohesion is actually evaluated at the "data structure" level rather than the "function" level.

Table 1: Attributes, Abstraction Levels, and Acceptance Criteria

- The documentation must be internally consistent (e.g., functional contracts must have a consistent format, design documentation must describe interfaces consistently).
- The discipline described in the documentation must be enforced (e.g., the code must represent consistent and correct application of the coding standards).
- The contracts must be consistent with the design documentation and the coding standards must be consistent with the software engineering discipline.

3 Modularity Attributes

The strength of the following six attributes are indicative of modularity and overall software quality:

- Code Cohesion
- Complexity
- Coupling
- Data Cohesion
- Duplicate Code and Data
- Extraneous Code and Data

Each of these attributes makes a unique contribution to the modularity of the system. Furthermore, a dependency exists among these attributes relative to the role they play in achieving a modular system. A basic goal of modularity as a software-engineering discipline is the minimization of complexity. A well-designed, simply constructed system is more easily understood than a system whose design and implementation are complex. Data and code cohesion and minimal coupling among modules contribute toward the goal of controlling complexity. Similarly, ensuring that no duplicate or extraneous code or data exist in the system helps to minimize complexity and to facilitate understanding.

3.1 Code Cohesion

Code cohesion is a measure of the strength of relationship between the activities performed by a software entity. Stevens [7] defined six categories of cohesion:

1. A module/function has *functional cohesion* if it performs activities related to a single purpose. Typically, a functionally cohesive module/function will transform a single type of input into a single type of output, will either work on only one type of variable or will move data from one type of variable to another. Examples of *functionally cohesive* modules are a stack manager and a queue manager. Examples of *functionally cohesive* functions are mathematical functions, an access check, and a dominance check.

    - *Functional cohesion* is the highest and most desirable form of cohesion.

2. A module/function has *sequential cohesion* if the output from each one of its functions/elements is input for the next function/element. An example of a *sequentially cohesive* module is one that contains the functions to write audit records and to maintain a running count of the accumulated number of audit violations of a specified type. An example of a *sequentially cohesive* function is one that transforms a label from an external form to an internal form, and then associates that label with an object (e.g., places it in the correct inode).

    - *Sequential cohesion* is a high form of cohesion.

3. A module/function has *communicational cohesion* if the functions/elements within it produce output for other function(s)/element(s) within it or use the output from other function(s)/element(s) within it. An example of a *communicationally cohesive* module is an access-check module that has the following functions: check the mandatory access, check the discretionary access, and grant access. An example of a *communicationally cohesive* function is a mandatory access check function that performs the secrecy access check and grants or refuses access based on the result.

    - *Communicational cohesion* is a moderate form of cohesion.

4. A module/function has *temporal cohesion* if the activities it performs need to be executed around the same time. It may have to operate on multiple types of input variables and/or may produce multiple types of output variables. Examples of *temporally cohesive* modules include initialization, recovery, and shutdown. An example of a *temporally cohesive* function is one that initializes a heterogeneous set of data structures.

    - *Temporal cohesion* is a low form of cohesion.

5. A module/function has *logical (procedural) cohesion* if it performs similar activities on different data structures. For a module, *logical cohesion* is present if the different functions are performing similar activities on different data types. For a function, *logical cohesion* is present if the elements are related only through common enclosing control structures such as *if ... then ... else*. An example of

a *logically cohesive* module is one where each function builds the integrity check information for a different object type (e.g., segment, file, device, page). An example of a *logically cohesive* function is one that manipulates different types of queues.

- *Logical (Procedural) cohesion* is a low form of cohesion.

6. A module/function has *coincidental cohesion* if it performs unrelated, or loosely related activities. If a module/function does not fall in any of the other cohesion categories, it has *coincidental cohesion.*

   - *Coincidental cohesion* is the lowest and least desirable form of cohesion.

Cohesion is an important software attribute in terms of understandability and maintainability. Code cohesion contributes to ease of understanding in that highly cohesive code performs a well-defined set of activities. Furthermore, localization of activities contributes to ease of maintenance.

Functional, sequential, and communicational cohesion are acceptable forms of cohesion. Coincidental cohesion is unacceptable. Temporal cohesion also is unacceptable except for the initialization code, recovery code, and shutdown code. Logical (procedural) cohesion is unacceptable at the function level, but acceptable at the module level.

3.2 Complexity

Complexity is a measure of how difficult a computer program is to understand (and thus to analyze and maintain). Minimizing complexity is the ultimate goal of a programming team's attempts to develop a system having good modularity characteristics. Controlling coupling and cohesion in the system contributes significantly to this goal.

Ever since Dijkstra [3] brought attention to the importance of clarity and elegance in programs (and "invented" structured programming), much attention has been focused on the advantages of structuring programs and data, of using high-level languages, of reflecting a program's structure in its written form (using spacing and indentation), and of top-down design. In spite of this attention, programmers still can produce programs that are difficult to analyze for correctness. Such programs, though written in a language that permits (or even encourages) well-structured programs to be produced, may be needlessly complex: they may communicate with other programs through side effects on global variables (a problem more of functional decomposition than of coding); they may contain portions that could be eliminated or combined with other portions if different algorithms or data structures were used; they may

contain unfamiliar expressions or constructions; or countless other factors may contribute to complexity. If the resulting complexity is too great, adequate assurance that the program is correct cannot be obtained.

A great deal of effort in the software engineering field has been expended in attempting to develop metrics to measure the complexity of source code. Most of these metrics use easily computed properties of the source code, such as the number of operators and operands, the complexity of the control flow graph, the number of parameters and global variables, and the number of levels and manner of interconnection of the call graph. [4] Some of these metrics have been used as the bases for commercial automated tools designed to measure complexity. The SAWG conducted a search for automated tools that might be useful in assessing the code complexity. Unfortunately, no tools capable of assessing the complexity of the kind of code generally found in the operating systems were identified.

The complexity of a system ultimately determines its understandability, maintainability, and testability. If the system is overly complex and difficult to understand, it will not provide the desired assurance that it works properly and securely.

Yourdon and Constantine [10] recognized several factors that affect the complexity of a computer program:

- The amount of information that must be understood correctly;
- The accessibility of the information; and
- The structure of the information.

In the absence of suitable automated tools for measuring complexity, the SAWG recommended a straight-forward (manual) analysis technique for measuring these complexity indicators: the size of modules and the time required to understand them. In other words, each module must fall within fairly specific size constraints (relative to numbers of statements), and the contract, design, and code must be comprehensible within the specified time periods.

3.3 Coupling

One must understand the interdependencies between the modules of a system in order to fully understand how the system works. *Coupling* is a term that encompasses how modules interact and how strong the dependencies are. Types of coupling include [5]:

- **Call Coupling** – Two modules are *call coupled* if they communicate strictly through the use of function calls. Examples of different types of call coupling are:

  - **Data Coupling** – Two modules are *data coupled* if they communicate strictly through the use of call parameters that represent single data items.
  - **Stamp Coupling** – Two modules are *stamp coupled* if they communicate through the use of call parameters which comprise multiple fields or have meaningful internal structure.
  - **Control Coupling** – Two modules are *control coupled* if one passes information which is intended to control the internal logic of the other.

- **Common Coupling** – Two modules are *common coupled* if they share a common system resource (e.g., variable).

- **Content Coupling** – Two modules are *content coupled* if one refers to the internals of the other in any way (e.g., modifying code of or referencing labels internal to another module).

Since modules are coupled to each other largely[4] through the actions taken through functions, the analysis must focus on functions and how they are coupled. Once the functions of the system have been examined, any instances of coupling that are intra-module can be exempted from further analysis, since the ultimate target of this analysis is inter-module coupling. Further, this analysis is not concerned with the precise strength of inter-module coupling, but rather focuses on making a binary choice. Hence, the result will be "acceptable" or "unacceptable."

Since the system architecture requirement of the TCSEC states that modules must be "... largely independent ...," coupling must be kept reasonably small. Some of the underlying reasons for this requirement are: largely independent modules are easier to understand and maintain (i.e., less complex); their use will cause fewer unintended side effects on other modules if something were to go wrong; and other assurance-determining efforts (such as penetration testing) may become more tractable.

Call coupling is always acceptable, and content coupling is always unacceptable. Common coupling is either acceptable or unacceptable, depending upon the outcome resulting from the application of the analysis method defined by the SAWG. The method involves ensuring that the scope of each variable is appropriate and analyzing the global variables. If a global variable is modified within a single module, but referenced by multiple modules, unacceptable common coupling is present. If a global variable is modified in a single function, the team must decide upon acceptability.

[4]It can be argued that modules sharing definitions, such as data structure definitions, are coupled. However, for the purposes of the analysis shared definitions are considered acceptable, though subject to the data cohesion analysis.

3.4 Data Cohesion

Statements that define data (such as data structure definitions, type declarations, variables and constants) must be analyzed in terms of data cohesion. *Data cohesion* refers to the strength of the relationship between a group of collocated statements that define data. Data cohesion applies to the module level and data-structure level.

Some of the terms defined in the section on code cohesion (see section 3.1) can be applied in this discussion to refer to types of data cohesion. Most types of data cohesion found will likely be categorized as *coincidental* (unrelated, or loosely related), *logical* (related in a logical sense) or *functional* (all contribute to a single goal; all are manipulated together).

A cohesion analysis of the data-defining statements within a system may be just as important as a cohesion analysis of the executable code. Both forms of analysis contribute toward the goals of understandability, maintainability, and testability. In addition, attention to data cohesion in the design of data structures encourages module independence. For example, data-defining statements may contain portions that are highly cohesive and therefore likely to be referenced from a single module, or from only a few, reducing coupling. An understanding of the major data structures should increase the understandability.

Data definitions must exhibit at least logical data cohesion. For example, the elements of a data structure or a group of data-defining statements packaged together should at least be logically related. Coincidental data cohesion is unacceptable, and functional data cohesion is preferred.

3.5 Duplicate Code and Data

The term *duplicate code* refers both to multiple instances of identical code and to multiple instances of code sequences that perform the same operation. The term *duplicate data* refers to multiple definitions of data structures that are (or could be) used interchangeably. Duplication can occur anywhere within the system; i.e., it is not limited to single functions or modules.

In some cases, code (or data) can be considered nearly duplicate. For example, two functions may be different only in that they perform an operation on different files. These functions may be replaced by a single function that accepts the filename as an argument, removing the duplication.

Duplicate code and data impose an unnecessary burden on system developers and maintainers, who must ensure consistency of all duplicate code and data in the system. Duplication can also increase the time required to understand the system. Efforts

should be made to eliminate duplicate code and data.

No duplicate code or data is acceptable.

3.6 Extraneous Code and Data

The term *extraneous code* refers to code in the system that serves no useful purpose in the evaluated product and, therefore, can be removed without affecting contract adherence. The determination of code necessity is performed at the code level. For example, functions that are never invoked, code that is circumvented by the execution logic, functions which simply return when called (and their associated calls), and segments of code that cause no effective state change in the system are all deemed extraneous.

Similarly, *extraneous data* refers to data structures, type definitions, etc. which can be removed without affecting a code's adherence to its contract.

Extraneous code and data hamper understanding of the system by serving as a distraction when attempting to analyze it. Extraneous code is also a problem in that maintenance or enhancements may invoke code which had been circumvented and, therefore, had gone unanalyzed and untested.

No extraneous code or data is acceptable.

4 Analysis

In order to meet the modularity criterion, each of these modularity attributes must be analyzed. The SAWG report [8] provides an analysis approach and acceptance criterion for each attribute. For example, the approach for analyzing code cohesion involves the development of a processing element flow graph [6]. In the case of coupling, the analysis involves two procedures: one (for content coupling) that is performed on all code analyzed and the other (for common coupling) that is performed only when other modularity attributes have identified a related problem area.

The modularity assessment is conducted during an architecture study involving several analyses: a preliminary design analysis, a preliminary code study, and a full code study. The purpose of the architecture study is to evaluate the design and implementation relative to the system architecture requirement; it is not to provide specific solutions or advice to the developers or to identify every instance of a discrepancy. The preliminary design analysis is conducted during the Vendor Assistance Phase of the Trusted Product Evaluation Program (TPEP) [9] and its purpose is to collect and examine evidence of the vendor's ability to produce the items required for the

preliminary code study.

The preliminary code study is conducted as early as possible in TPEP, during either the Vendor Assistance Phase or the Design Analysis Phase. Its objective is to gain assurance that the final product will meet the modularity requirements. The assumption is that if the vendor appears at this point to understand the implications of the modularity requirements relative to both design and implementation, and if the code developers are adhering to the coding standards, the final product will be "modular." The full code study is conducted as early as possible during the Formal Evaluation Phase of TPEP and is the activity that ultimately determines whether the system meets the modularity requirement.

The methodology developed by the SAWG has been used by a few teams in conducting their architecture studies of products under evaluation for class B2 ratings. Feedback from these teams was used in refining this process. As the methodology is applied to more evaluations, it will be further refined and revised.

5 Summary

Meeting the modularity criterion of the System Architecture requirement is critical to obtaining a class B2 rating. The SAWG report identifies six attributes as important in meeting this criterion, provides a methodology for evaluating these attributes, and discusses the evidence required to support the assessment.

Acknowledgments

The authors wish to thank Steven LaFountain, Patricia Moreno, Joseph Bulger, and Mario Tinto for their contributions to the development of this paper.

References

[1] *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985.

[2] Baer, J-L. *Computer Systems Architecture*, Computer Science Press, 1980.

[3] Dijkstra, E.W., "Programming Considered as a Human Activity," *Proceedings of the 1965 IFIP Congress*, 1965, pp. 213-217.

[4] Fairley, R., *Software Engineering Concepts*, McGraw-Hill, Inc., 1985.

[5] Martin, J. and C. McClure, *Structured Techniques for Computing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985.

[6] Ott, Linda M. and Jeffrey J. Thuss, "The Relationship between Slices and Module Cohesion", *Software Engineering, Proceedings of the Eleventh International Conference*, Pittsburgh, PA, May 15-18, 1989, pp. 198-204.

[7] Stevens, W. P., *Using Structured Design*, John Wiley and Sons, 1981.

[8] *Trusted Computer System Architecture: Assessing Modularity*, National Security Agency, Ft. George G. Meade, MD, in press.

[9] *Trusted Product Evaluations: A Guide for Vendors*, National Computer Security Center, NCSC-TG-002, 22 June 1990.

[10] Yourdon E. and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice-Hall, Inc., 1979.

COMPANION DOCUMENT SERIES TO THE TRUSTED DATABASE MANAGEMENT SYSTEM INTERPRETATION[1]

LouAnna Notargiacomo
Victoria Ashby
Vinti Doshi
Jarellann Filsinger
Sushil Jajodia[2]

The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102

Lieutenant Colonel Ron Ross, USA

National Computer Security Center
9800 Savage Road
Fort George G. Meade, MD 20755

Abstract

In fiscal year 1990, The MITRE Corporation was tasked by the National Computer Security Center (NCSC) to begin the development of a series of companion documents to the *Trusted Database Management System Interpretation (TDI) of the Trusted Computer System Evaluation Criteria.* During fiscal years 1990 and 1991, four TDI companion documents were developed discussing inference, aggregation, referential integrity, and auditing. This fiscal year, two additional documents are under development dealing with the topics of high-assurance discretionary access control (DAC) and polyinstantiation. This paper presents an overview of these companion documents.

Background

The *Trusted Database Management System (DBMS) Interpretation (TDI) of the Trusted Computer System Evaluation Criteria (TCSEC)* [1], published by the National Computer Security Center (NCSC), provides evaluation guidance for trusted systems that are composed of parts. An example of such a system is a trusted DBMS that runs on a trusted operating system. While the title of this document refers specifically to DBMSs, a conscious decision was made by the NCSC to deal with only the aspects of trusted DBMS evaluation that apply to all trusted applications designed to run on a trusted operating system. Therefore, the TDI does not deal with many DBMS-unique issues. This decision was made for two reasons. The first reason was the desire to have this interpretation apply to a wide range of trusted applications. The second reason was that, in the extensive discussions of DBMS-specific issues that arose during the TDI's development, it became obvious that many of the issues were still open research topics. As a result, it was decided that it would be inappropriate to define evaluation guidance in a technology area that was

---

1 Sponsorship of this work is by the National Computer Security Center under contract DAAB07-91-C-N751.

2 Also affiliated with the Center for Secure Information Systems and the Department of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030-4444.

undergoing rapid change. Applying such guidance would both stifle research innovation and run the risk of endorsing an unproven approach.

Nevertheless, since trusted DBMS evaluations were expected to begin when the TDI was published, it was recognized that information on DBMS-specific trust issues still needed to be disseminated to the security community at large. Therefore, the decision was made to develop a series of companion documents to the TDI covering DBMS-specific issues.

MITRE's Role

In fiscal year 1990, MITRE was tasked by the NCSC to begin the development of the TDI companion document series. The development plan for these documents is that for each topic area, MITRE will develop a draft document to be reviewed first by the Government and then released for public review. Comments from these reviews will be incorporated into the documents before final NCSC approval and publication.

Each TDI companion document is devoted to a particular DBMS-specific research area. These documents discuss the research problem, present an overview of relevant research and development work, in some cases include additional problem analyses, present any appropriate conclusions drawn from the analyses, and summarize any additional work needed to resolve the problem. These documents are intended to be published as Technical Reports. Rather than serve as evaluation guidance, they will instead serve to disseminate information on the current state of the art for each topic. In contrast to evaluation guidance, which represents a consensus position, these Technical Reports deal with evolving technology, and thus some controversy over their content is expected.

During fiscal years 1990 and 1991, four TDI companion documents were developed on the following topics: inference, aggregation, referential integrity, and auditing. Draft versions of the inference, aggregation, and referential integrity documents have been delivered to the Government for review and should be available soon for external peer review. A draft of the audit companion document is near completion for delivery to the Government. This fiscal year, two additional documents are under development dealing with the topics of high assurance discretionary access control (DAC) and polyinstantiation.

Overview of the TDI Companion Documents

The purpose of this paper is to present an overview of the technical content of the companion documents that MITRE has developed to date and a brief discussion those documents currently under development. For each companion document, the problem being addressed is described, an overview of the document content is given, and any conclusions are summarized.

Inference Companion Document: *Inference Problems in Multilevel Secure Database Management Systems*

Using a DBMS, users can draw inferences from the information they obtain from a database. The inference could be derived purely from data obtained by querying the DBMS, or it could additionally depend on some prior knowledge obtained from outside the database system. An inference becomes a problem when more highly classified information that the user is not authorized to access can be inferred from less classified information.

Many difficulties are associated with determining when more highly classified information can be inferred from less classified information. The biggest problem is that it is impossible to determine

precisely what a user "knows." The problem is at least manageable if the closed-world assumption can be adopted. Under the closed-world assumption, if information *Y* can be derived using information *X*, then both *X* and *Y* are contained in the database. By ruling out inferences that lie outside the database, the closed-world assumption provides a structured framework within which inference problems can be studied [2].

The scope of the inference problem, as addressed in the inference companion document [3], is limited in several ways. To stay in the mainstream of database security work, the report is limited to the relational model, although other models have occasionally been addressed in research literature [4, 5]. Also, this report omits consideration of inference control for statistical databases since this topic has been well covered elsewhere.

Inference problems can be grouped into three classes. The first class consists of inferences that can be derived from the low data retrieved from the database by a user, for example, if a user is allowed to issue queries conditioned on data that are supposed to be invisible to the user. The second class of inference problem involves inferences that can be derived from the low data together with the metadata (for example, in the form of integrity or value constraints) stored in the database. Inference problems can be caused by types of key integrity constraints, functional and multivalued dependencies, and value constraints. The third type of inferences are those that require, in addition to the information retrieved from the database, some external knowledge. This external knowledge usually consists of general knowledge about the environment, knowledge of how information is logically connected to allow an inference to be made, or an understanding of algorithms that can be applied to the lower level retrieved data and used to compute higher level data values.

To control unauthorized disclosures due to inference, several approaches have been proposed in the research literature. These approaches are briefly addressed below:

- Limiting the data used to satisfy a query to data objects the user is cleared to access
- Allowing polyinstantiation by including the object's sensitivity level in its primary key
- Raising the classification of data items that may be used to infer higher level data
- Raising the classification of value constraints to the level of the data that can be inferred using the constraint
- Using history data recorded in the audit trail to identify inferences derived over multiple actions
- Allowing limited inferences under controlled conditions

The inference companion document concludes that although many methods have been developed for dealing with inference problems, elimination of undesirable inferences remains very much an intuitive process. To date, a majority of research on the inference problem has concentrated on the database design phase and not on the identification of inferences that occur dynamically during command execution. Application of the methods discussed for the identification and elimination of inferences during database design can yield very good or very bad results, depending on the skill of the database designer. Since the elimination of all inference problems is an extremely complex task, this report provides additional guidance on the vital issues involved in the elimination process.

Aggregation Companion Document: ***Aggregation Problems in Multilevel Secure Database Management Systems***

For the purposes of the aggregation companion document [6], the aggregation problem refers to the direct association of data that results in a higher classification level than the session level of the user producing the aggregate. That is, the true classification level of the aggregate is higher than the least upper bound of any input used to create the aggregate. This definition does not reflect inclusion of any data outside the data model, or knowledge the user may possess that is outside the data model.

The definition of the aggregation problem is further refined to include both *cardinality aggregation* and *data association* problems. In cardinality aggregation, the label on the data aggregate is dependent on the number of like pieces of data. A data association problem occurs whenever two values seen together are classified at a higher level than the classification of either value individually. The data association problem can be differentiated from the cardinality aggregation problem in that what makes the information sensitive is not the aggregate of the two lists, but the exact association between the pairs of data objects.

While the policies for mandatory and discretionary access control can be implemented using generalized mechanisms that are application independent, no such relationship between aggregation control policies and implementation mechanisms is known. The data aggregation problem is specific to each database and the environment in which it exists, so it cannot be automatically handled. Aggregation relates to the semantic structure of data; aggregation control is, therefore, data dependent and must be tailored to the data. Thus, aggregation control is a problem inherent in the data being managed, not a problem inherent in the data management software of the DBMS (although DBMS mechanisms can be used to control this problem).

In the aggregation companion document, a distinction is made between aggregation control techniques that are useful in single-level (or system-high) database applications and control methodologies useful in multilevel applications. In addition, techniques that can be used during the database design phase are distinguished from tools intended to be used during application execution.

In single-level database applications, the controls that have been used are for the most part operational controls, for example, the control and review of hard copy output. Those that are not operational divide into two groups: those used during the database design phase before the database is operational, and those used while it is operational but only after database access has occurred. Although research has been done on tools that detect aggregation problems as the database access occurs, the performance impact of tools and techniques developed to date may be too severe to allow such tools in an operational system, even when the tools prove useful.

Data aggregation can be controlled through the database design process by using the designer's knowledge about the data and how the data can be aggregated. For example, a decision can be made to omit data from the database that would combine with other data to form an aggregate above the system-high level. Another database design technique for aggregation control divides data that could aggregate into separate structures. In single-level database applications, discretionary access controls have been applied to the separate structures, preventing their combination by most users. For relational databases, these discretionary access controls could be associated with predefined views. (A view is a named query statement that identifies a subset of the database.) To make the use of views for discretionary access control effective requires restricting queries to only those that execute against predefined views. Audit trail analyses may

also be used for aggregation control. Assuming sufficient data has been audited, audit trail analyses can be automated to alert the security officer to any possible aggregation problem.

For multilevel databases, the aggregation companion document covers both pragmatic control approaches and a range of proposed approaches in database security research. The pragmatic control strategy discussed deals with data aggregation control in a database implemented by an MLS DBMS, in which both predefined and ad hoc queries are allowed. With this approach, the appropriate label for predefined, standard queries is defined and set. All ad hoc queries, however, are labeled at the system-high level, independent of the user's session level. In addition to query restriction, the other control strategies presented for single-level databases have also been suggested for multilevel databases implemented using an MLS DBMS. However, all are restrictive.

The aggregation companion document then reviews several approaches that have been pursued in DBMS security research for multilevel databases. The SeaViews project uses the concepts of classification constraints and aggregation constraints [2]. However, according to a more recent paper on the SeaViews project [7], results show that a proper combination of database design and access controls will control many instances. In [8], the Brewer-Nash Chinese Wall model is extended to address aspects of the data aggregation problem. With this model, datasets are grouped into "conflict of interest classes" and by mandatory ruling all subjects are allowed to access, at most, one dataset belonging to each such conflict of interest class. Based on this model, Meadows derived a lattice-based information flow policy that allows the construction of a system which prevents users from accessing aggregates they should not be able to see. The ASD-Views project uses an approach in which data accesses are constrained to go through relational views. Associated with each view definition is a label that reflects the level of the aggregate response [9, 10]. In the Lock Data Views (LDV) project, the basic security policy is extended to incorporate name-dependent, content-dependent, and context-dependent classification policies, as well as inference control [11, 12, 13]. With this approach, aggregation constraints are defined and then enforced after the query results have been built. Finally, in [14] a security algebra is defined that can be used to identify the intersection between sets of items that when combined result in a higher level aggregate. The data items at the intersection are then labeled at the aggregate level.

The aggregation companion document concludes that to handle data aggregation, the DBMS must supply mechanisms, or tools, for data aggregation control. However, no one mechanism currently exists that will completely solve the data aggregation control problem, even when restricted to information contained in the database. Instead, a group of tools exists that can be considered for use in each specific environment dependent on individual applications. These mechanisms by themselves, are not sufficient for aggregation control; how the mechanisms are used is most important.

Entity and Referential Integrity Document: *Entity and Referential Integrity in Multilevel Secure Database Management Systems*

The Entity and Referential Integrity companion document [15] focuses on the problems associated with the enforcement of the relational entity and referential integrity constraints defined in the current American National Standards Institute (ANSI) standard for the SQL2 language (the current standard for the relational database data manipulation and data definition language) [16], when applied to a multilevel database. These SQL2 features are designed to allow rules to be defined that, when enforced, ensure that the relationships between data objects are not invalidated as a result of data insertion or modification. In a multilevel database, these relationships potentially exist between objects at different sensitivity levels. Development of a relational DBMS that meets these SQL2 requirements would include features to update related data objects automatically,

maintain database integrity, or prevent an update that might result in a database integrity violation. Unfortunately, enforcing referential integrity rules between objects at different sensitivity levels may permit the signaling of information between users operating at different sensitivity levels. The objectives of the entity and referential integrity companion document are to analyze the ways of enforcing referential integrity controls in MLS DBMSs, and to identify those referential integrity rules that can be enforced without compromising secrecy.

The entity and referential integrity companion document first defines these forms of integrity with respect to single-level DBMSs and describes various concepts related to referential integrity. It then defines the basic concepts of entity and referential integrity with respect to multilevel database relations and gives a detailed analysis of referential integrity in the multilevel context. The analysis considers different instances of the relationship between the access class of the foreign key and the access class of the referenced primary key, both under different levels of granularity of the labeled objects and with or without polyinstantiation.

The extension of the concepts of referential integrity from single-level relations to multilevel relations is not straightforward. This complexity arises because restrictions are needed to provide referential integrity control in MLS DBMSs without compromising secrecy. The basic requirement for referential integrity is that each referencing foreign key value must have an identical target primary key value in the referenced relation. An additional requirement for multilevel relations is that the foreign key and the primary key should be uniformly classified (i.e., all attributes included in the key should have the same access class).

The entity and referential integrity companion document concludes that enforcing referential integrity when the access class of the foreign key is equal to the access class of the referenced primary key is simple and without any ambiguity. All integrity rules apply in this case, whether or not the relations allow polyinstantiation. In fact, when polyinstantiation is allowed, the access class of the primary and foreign key values must be included as part of the key to disambiguate references and allow the referential integrity rules to be enforced. In the second case, however, when the access class of the foreign key does not dominate the access class of the primary key, referential integrity completely fails. In the final case in which the access class of the foreign key dominates the access class of the referenced primary key, some of the integrity rules apply when the action is to be taken on deletion or modification of a key value. The exact action differs dependent on the granularity of the labeled data object. The entity and referential integrity companion document concludes with a table that enumerates the various cases and the conditions under which they can be applied.

Auditing Companion Document: *Auditing in Multilevel Secure Database Management Systems*

The auditing companion document discusses auditing in Trusted DBMSs (TDBMSs). First, the objective of auditing is reviewed as it applies to trusted data management. The characteristics of auditing in a TDBMS and a trusted operating system are compared. The primary differences between the two stem from the variety and complexity of DBMS object structures and the methods used to manipulate them. Unlike operating system objects, TDBMS objects and their metadata have semantic interrelationships that are the basis for data manipulation. An examination of the TDBMS object structure and database semantics is undertaken to help define the scope of TDBMS auditing.

Another difference between DBMS and operating system auditing concerns the need to audit actions that impact data integrity. Although the TCSEC is concerned with integrity when it directly

affects the system's ability to maintain confidentiality (i.e., integrity of the Trusted Computing Base and sensitivity labels), it does not discuss auditing of actions that might impinge on the integrity of application data. However, an important characteristic of a TDBMS is its support for preserving the integrity of the data it manages. This report recommends extending the scope of auditing for TDBMSs to support this essential capability.

The types of operations that can be carried out on TDBMS objects and metadata are examined to provide examples of actions to be audited. The SQL2 language provides a framework in which to analyze the audit implications on the different types of database objects and the actions that can be performed on these objects. An appendix lists specific recommendations for the types of information that should be audited when executing each of the basic types of SQL2 statements. Additionally, the need is recognized for auditing of system-level TDBMS actions, such as database utilities and concurrency control. Finally, a number of remaining open issues and research areas related to TDBMS audit are identified.

Polyinstantiation Companion Document: ***Polyinstantiation in Multilevel Secure Database Management Systems***

Work on the development of a TDI companion document on polyinstantiation began in fiscal year 1992. This document will first present an overview of the basic security problems in multilevel databases caused by entity and key integrity and describe methods for applying polyinstantiation to resolve these problems. Next, the problems with data integrity that are caused by polyinstantiation will be discussed and examples using different object granularities will be presented. The document will provide a survey of the various approaches being used by MLS DBMS vendors, followed by an overview of the approaches being proposed in the research literature. Finally, the problems that still need resolution will be identified.

High Assurance Discretionary Access Control Companion Document: ***High Assurance Discretionary Access Control in Multilevel Secure Database Management Systems***

The scope of the high assurance discretionary access control (DAC) companion document is to analyze the technical problems limiting the evaluation of view-based DAC capabilities at higher than the B1 evaluation level. The two main areas that will be addressed in this document are the increase in TCB size and complexity when the view-based DAC approach is used, and the problems involved with developing and proving a formal model of view-based DAC, including a discussion of the DAC Trojan horse problem.

Summary

The TDI companion document series supports the dissemination of information concerning database security areas pertinent to the development of MLS DBMS products, the evaluation of application requirements, the analysis of product capabilities, and the evaluation of trusted DBMS products. The release of the Inference, Aggregation, and Referential Integrity documents for peer review is intended to occur shortly, followed by the release of the other documents as they are completed. The availability of this information will further encourage technical discussion, research, and technology advances to address these critical and difficult problems.

References

1. Department of Defense, *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*, National Computer Security Center, NCSC-TG-021, Version 1, April 1991.

2. Denning, Dorothy E., "A Preliminary Note on the Inference Problem in Multilevel Database Management Systems," *Proceedings of the National Computer Security Conference Invitational Workshop on Database Security*, Ft. Meade, MD, June 1986.

3. Jajodia, Sushil, *Inference Problems in Multilevel Secure Database Management Systems*, DRAFT, The MITRE Corporation, McLean, VA, June 1992.

4. Berson, Thomas A., and Teresa F.Lunt, "Multilevel Security for Knowledge Systems," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, April 1987.

5. Patkau, Burton H., and David L. Nennenhouse, "The Implementation of Secure Entity-Relationship Databases," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, April 1985.

6. Ashby, Victoria, *Aggregation Problems in Multilevel Secure Database Management Systems*, DRAFT, The MITRE Corporation, McLean, VA, June 1992.

7. Lunt, Teresa F., "Aggregation and Inference: Facts and Fallacies," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1989.

8. Meadows, Catherine, "Aggregation Problems: A Position Paper," *Proceedings of the 3rd RADC Workshop*, 1990.

9. Hinke, Thomas H., "Inference and Aggregation Detection in Database Management Systems," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, April 1988.

10. Wilson, Jackson, "Views as the Security Objects in a Multilevel Secure Relational Database Management System," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, April 1988.

11. Haigh, J. T., R. C. O'Brien, P. D. Stachour, and D. L. Toups, "The LDV Approach to Database Security," *Database Security: Status and Prospects, Proceedings of the IFIP Working Group 11.3 on Database Security*, North-Holland, September 1989.

12. Haigh, J. T., R. C. O'Brien, P. D. Stachour, D. L. Toups, and B. M. Thuraisingham, *Secure Distributed Data Views: Final Technical Report for a Database Management System, Volume 6: Deficiencies Analysis*, Honeywell Systems and Research Center, May 1989.

13. Stachour, P.D., and B. M. Thuraisingham , "Design of LDV: A Multilevel Secure Relational Database Management System," *IEEE Transactions on Data and Knowledge Engineering*, Vol. 2, No. 2, June 1990.

14. Lin, T.Y., "Commutative Algebra and Aggregation," *Proceedings of the 2nd RADC Database Security Workshop*, 1989.

15. Doshi, Vinti, and Sushil Jajodia, *Entity and Referential Integrity in Multilevel Secure Database Management Systems*, DRAFT, The MITRE Corporation, McLean, VA, June 1992.

16. American National Standards Institute (ANSI), *(ISO/ANSI working draft) Database Language SQL2*, J. Melton, editor, *ANSI X3H2-90-309*, August, 1990.

COMPUTER SECURITY
and
TOTAL QUALITY MANAGEMENT

Major Gregory B. White, USAF Academy, CO
Mr. Lee Sutterfield, AFCSC/SRO, San Antonio, TX
Mr. Chuck Arvin, CTA, Colorado Springs, CO

ABSTRACT

In recent years, a number of articles and books have been written on the importance of quality in the manufacturing of a product. In fact, the federal government has recently launched its own quality program known as Total Quality Management and published a series of booklets under the title of the Federal Total Quality Management Handbook outlining the program. While at first it may seem that the idea of infusing quality into any product or process is a simple matter, it has in fact been shown to be very difficult. How this new push for quality affects, and is affected by, security is the subject of this paper.

INTRODUCTION

The implementation of effective computer security policy and procedures in the field has always been difficult. The practical concerns of computer system administrators and users for daily production usually places security concerns as a low priority. The result is often a poor security posture despite extensive education and awareness programs and easy availability of a growing set of effective security products and tools. The field is not lacking in security policies and procedures yet security officers still find it difficult to get the many users of their systems to follow the established security practices. What then can be done to bring about a sound security posture? The premise of this paper is that what is lacking is a clearly defined process for security and a means of continually refining that process for the better. The answer to the computer security problem lies not in technology or even clear policy. The answer lies in the use of Statistical Process Control (SPC) and Total Quality Management (TQM).

This paper will focus on the application of Total Quality Management to computer security. Currently the most recognized application of TQM and SPC has been within the manufacturing environment, usually as part of the approach to management known as Total Quality Control. In these traditional applications, SPC has been applied to what can be termed as "wholly owned environments" where organizational structures fall under a common authority. More often in computer security the responsibility for security has been delegated to an office which falls outside of the traditional organizational line of authority making it hard for the security officers to exert any real influence at all. With the introduction of SPC and TQM into the workplace, many of the obstacles now faced can be either reduced or eliminated.

COMPUTER SECURITY AS A PROCESS

The theory behind Total Quality Management has its roots in the work of several individuals, the most notable of which is Dr. W. Edwards Deming. Dr. Deming has been

credited with the revitalization of the Japanese industry after World War II. His ideas, which are considered by some to be unorthodox, in turn find their roots in the work of earlier statisticians and are based on the control of a process through statistical monitoring methods. He believes that quality doesn't cost, it pays and can be obtained by the application of statistical control methods to each step of a manufacturing process. This differs from traditional quality control methods in this country which have been historically applied at the point a finished product comes off of the assembly line. The problem with the latter approach is that the defective item rejected at the end of the line may be the result of a problem early on in the manufacturing process. Everything done to it along the line was a waste of time since every step was working with a defective item. Had the quality of the product been monitored at all steps of the process the defective item would have been detected and rejected early on and subsequent steps would not have needlessly been accomplished. Additionally, the monitoring of the entire process also has the advantage of allowing a company to identify certain steps that may be causing an inordinate number of defective products and allow management to take steps to fix these problems. While this is a tremendous simplification of the methods taught by Dr. Deming, it should provide the basis from which the application of TQM to computer security can be discussed.

The traditional sequence for the manufacturing of a product followed three basic steps. These steps, as illustrated in figure 1., were: 1) Design the product, 2) Make it, and 3) Try to sell the product.

This process is described both by Dr. Deming [1] as well as others, such as Mary Walton [4], in books written about Dr. Deming's methods.

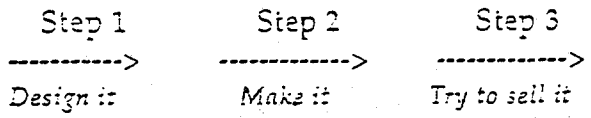
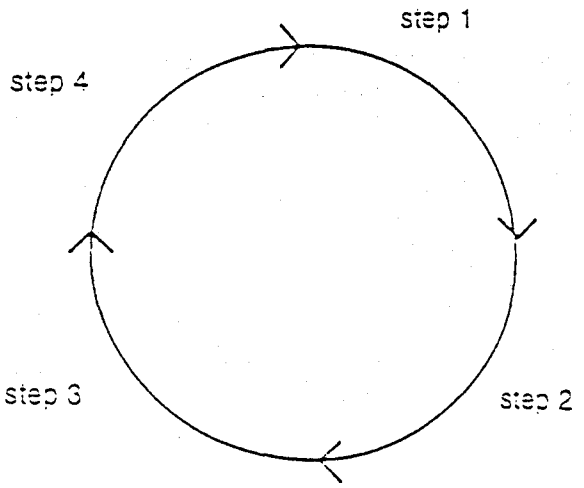


Figure 1.

Dr. Deming, following the works of an earlier statistician, Walter A. Shewhart, proposes a change in this traditional process flow to provide feedback and make the process flow more responsive to the needs of the customer [1]. This new way of "doing business" is referred to as the Shewhart Cycle and consists of the four steps as shown in figure 2.

There are two notable changes to the process from that depicted in figure 1. The first change is in the second step. Now, instead of just producing the product, this step also includes a testing of the product. The other change is in the addition of a fourth step which is designed to determine the acceptance of the product by the consumers and to provide feedback to the design process for the next generation of the product. A more subtle difference is in the acceptance that any product is affected by previous designs and products. In fact, every product is affected by a myriad of intermediate steps in the process. When we view every step of a larger manufacturing process as an individual process itself, it is interesting to note that SPC can be applied to each of these steps individually. This last fact is crucial to understanding how SPC can help improve an organization's security posture.

The Shewhart Cycle



step 1: Design the product
step 2: Produce it, test it during production
step 3: Market it
step 4: Receive feedback from the users/consumers

Figure 2.

Another subtle difference in these two approaches occurs in the fourth step. This step, where the feedback takes place, is crucial because it acknowledges that the user is important (in fact the most important person). The user's needs/desires are what should drive the design of the product. In essence, we are asking "What is it that the user needs and what is it that our product is delivering?" While this may seem to be the most logical and obvious way to approach business, it has not been the way manufacturing and industry have approached business in this country. Instead, a product is developed and the company then proceeds to try and convince the consumers that it is something they can't live without.

In his book, Out of the Crisis [1], Dr. Deming split companies into two broad categories: manufacturing and service. In other words, any company is providing one of two things; a tangible product to be sold or a service to be rendered. Computer security falls under the second category as a service to be provided to an organization. Accepting this and the idea that any portion of a process can be broken down into smaller sub-processes, leads to the question "What does the computer security process entail?" In the past, computer security followed a three step process similar to the one found in figure 1. In this case the steps revolved around the security regulations or directives that individuals were required to follow. This traditional computer security process is depicted in figure 3.

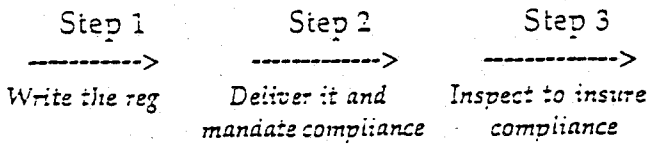
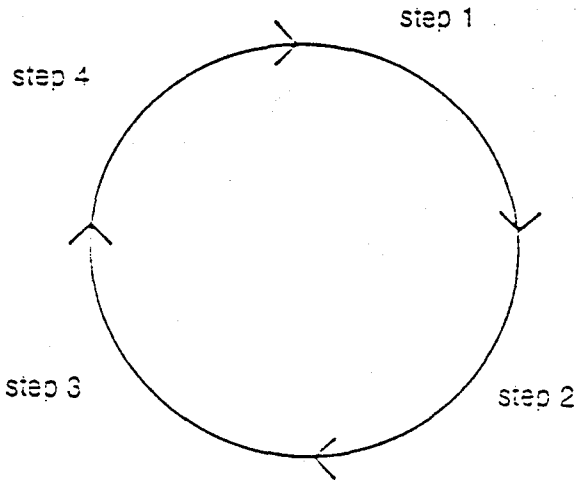


Figure 3.

Just like the three step process used in manufacturing found in figure 1, by handling computer security in this manner we are setting our "workers" up to fail for a number of reasons to be detailed in the next section of this paper.

This process, like the earlier one for manufacturing, has to be changed to bring into the process the comments of those who are in the best position to detect any flaws in the existing process. This new process, as depicted in figure 4, is modelled after the Shewhart Cycle.



step 1. Write the reg and develop tools
step 2. Deliver the regs and the tools to implement them
step 3. Market the service
step 4. Monitor effectiveness of regs and tools and obtain feedback

Figure 4.

In this new process, the goal of providing security for the information stored on a computer is the same; it is the process of how it is done that is changed. Since it is most likely the individuals in the field who will have the best idea about not only how to circumvent the controls in place but also how to improve on them, they become an integral part of the process. In addition, not only are the regulations provided to the individuals in the field but also a series of tools that will help them implement them. These tools in turn are subject to continual updating and revision as inputs are received from those who use them.

Feedback, as is shown, is an important part of this new process. No longer can the security policy makers afford to operate in isolation from those who implement the policies developed. The policy makers, however, can't afford to wait for feedback to filter up the chain, they have to actively seek out the comments from those in the field. Often this is the most difficult task of managers, especially as an organization attempts to change from the traditional three step process to the Shewhart cycle.

Another problem in security is actually defining our customer and our product. Defining our customer as any individual who uses a computer system (i.e. "Security is everyone's business") may make for a good poster but is not really realistic. Instead, our customer should be any individual who has, as part of their job, direct security responsibilities. This narrows our scope but doesn't lessen the responsibility since we will still be concerned with delivering the tools to our customers so they can do their job (which entails interacting with the users). The product, a service, is not security regulations, policy, and tools but rather good security posture. While this at first may seem to be somewhat nebulous, it in fact is what we are after. We are not actually concerned with whether security regulations are produced, we are concerned with the security of our systems and the information they process. Defining our product as good security posture provides us with a measurable and meaningful goal and measurable characteristics. A measurable process is essential to being able to apply statistical process control. In

addition, a better defined process helps to develop well defined sub-processes and clearer lines of authority.

As an organization attempts to change from the old style to the new style, there are a number of very basic changes that will need to be made. Dr. Deming has developed a list of 14 points which he considers essential in changing to a process oriented organization. These 14 points just as easily can and should be applied to computer security.

COMPUTER SECURITY AND DEMING'S 14 POINTS

The 14 points that Dr Deming developed have been updated and modified continually over the last 40 years as he has helped businesses in several countries restructure the way they operate. These 14 points are the management side (TQM) of the philosophy with statistical methods (SPC) being used to monitor the process. This management side of the philosophy is really a mindset or commitment to total quality. It is important to note that in order for the 14 points to work there has to be an almost universal commitment to them from top management on down. This is emphasized in the first of the 14 points which is:

1. Create constancy of purpose for the improvement of product and service.

This point goes right to the heart of the Deming method. In a business, whether the object is manufacturing a product or providing a service, it results in the acknowledgment that the customer is the most important part of the process. In computer security, as in other service oriented organizations, what is essential is the delivery of a quality product to keep the customer satisfied. The problem is to get everybody working together towards this end. Security will not work if only the few security personnel are concerned. It will only work if everyone is concerned and is actively working toward a secure environment. This leads to the second of the 14 points.

2. Adopt the new philosophy

For this philosophy to truly work, the entire organization must acknowledge that quality is the most important consideration. Before this approach, the norm was to be concerned with some standard such as X units produced every day or no more than Y defective products. The new philosophy is not concerned with numbers but rather with the quality of the items produced. In a service organization, such as computer security, the motivation should be that no unauthorized individual gain access to the organizations resources in order to obtain information or disrupt services. The emphasis in the field should not be on a blind implementation of security policies and regulations but rather on the securing of the computers and their data. This may mean going beyond what is required in the regulations or it may even mean doing something else and then providing feedback on a needed change to a regulation. For the new philosophy to work, the organization can't afford to have individuals hiding behind regulations. The new philosophy needs individuals who are actively looking for ways to improve the process. One way that the effectiveness of a product or service has been checked in the past was through the widespread use of inspections which is addressed in the next point.

3. Cease dependency on inspection to achieve quality

Dr Deming has stated "Inspection to improve quality is too late, ineffective, costly. When [a] product leaves the door of a supplier, it is too late to do anything about its quality. Quality comes not from inspection, but from improvement of the production process." (1) Dr. Deming is saying that if we wait until the end to check for defects it is too late, nothing can be done about it at that point. Instead we should monitor the process to look for ways of preventing the defect from happening. In computer security we too often rely on inspections to give feedback on how well an organization is complying with regulations and assume that if they did well on an inspection then their security must be good. Conversely, if they failed the inspection then they are doing poorly. What results is either a possibly unwarranted feeling of security or an emphasis on the area that caused the poor inspection at the expense of others. While inspections can serve a useful purpose, care must be exercised to insure that they don't become the main focus but rather a tool to ***monitor the process.***

4. End the practice of awarding business on the basis of price tag alone.

A more expensive, quality product is often times more economical in the long run than a lower priced product. This equates to the acknowledgment in computer security that, while security may cost, in the long run it may mean that the piece of information that might give a competitor the edge in a contract, or cause the loss of a pilot in combat, remains secure. There is an overhead associated with security. It would be nice if we didn't have to spend money on a safe to store information in, but we understand the world we live in necessitates it. We also accept the fact that if we purchase poor quality locks we put our valuables in jeopardy. We grow up in a society which secures its valuables, we are trained from childhood to understand this, but computer security is new and not well understood. It, however, is no less important.

5. Improve constantly and forever the system of production and service.

Quality is not something that is added on later, it should be emphasized starting in the design phase. This is not only true when building a car but also when discussing security. It is not static either. The environment and threat changes constantly. Security needs to be constantly evaluated to ensure that it is still adequate. In addition, as a result of the feedback loops in the process, every new system that is installed should be more secure than the previous one.

6. Institute training (and retraining).

Kaoru Ishikawa stated "Quality Control begins with education and ends with education."(2) The same can be said about security; it begins with education and ends with education. This is the single most important point in computer security. All too often a worker is trained by another worker. This has been shown to be ineffective as any misconceptions or misunderstandings about the job are transferred from the old worker to the new one. In computer security we have found another ineffective method of training: basing our training on regulations. Our training

---

1) Deming, W. Edwards, Out of the Crisis, Massachusetts Institute of Technology Center for Advanced Engineering Study, 1989, pg 28.
2) Ishikawa, Kaoru, What Is Total Quality Control? The Japanese Way, Prentice-Hall, Inc, Englewood Cliff, NJ, 1985, pg 13.

should be data oriented which means not teaching regulations and directives but teaching about our security posture and how to improve it. You can't just hand an employee a manual or directive and expect them to fully understand what is written. People are different. They learn differently and may interpret the same passage of text differently. You also can't expect to give a security briefing once a year and have anybody take it seriously. Training must be formal and fully supported in order for it to work.

7. Adopt and institute leadership.

In this new philosophy, the supervisor has an increased importance. "A supervisor must be more than a judge or overseer as the name implies. In this new economic age, he must be a coach and a teacher. The prime responsibility of a supervisor must be to develop his people so they continually improve, so they can do a better job." (3) Management can't just supervise or monitor but should be leading the quest for quality.

8. Drive out fear.

It is important in any process, and crucial in computer security, to drive out any fear the workers may have. Nobody can do their best until they feel secure in their job and environment. Fear not only impairs an individual's performance but also causes the "padding" of figures so perceived adverse effects will not occur when the process seems out of line. This is especially true in security which traditionally has taken a punitive point of view. In the future, we should emphasize the collection of data in a non-punitive roll--make security a team effort. In addition, too much security can strangle an organization and adversely affect its output. "Another loss from fear is inability to serve the best interests of the company through necessity to satisfy specified rules, or the necessity to satisfy, at all costs, a quota of production." (4) Fear changes the product from "improve the security posture" to "produce reports and comply with regulations," a subtle but crucial change.

9. Break down barriers between staff areas.

Simply put, this means that the lines of communication must be opened between management, people in R&D, designers of the tools used, and the users of the product. If there is no communication (no feedback), tools will be used which do not meet the needs of the field. When this happens, it dooms those using them to failure since they have "defective" tools to work with. This in turn affects their attitude toward the job generally resulting in dissatisfaction. Instead, when the tools are designed with inputs from those who will use them, the likelihood they will be used increases dramatically. This not only applies to tools designed to aid a security officer but also to security as part of a larger process. Too often in computer security we don't have a good understanding of what the overall product is of the organization and the frontline managers don't understand how security (or lack of security) can effect the product. Communication between the security managers and the operations managers is essential.

---

3) Scherkenbach, William W., The Deming Route to Quality and Productivity, CEEPress, Washington D.C., 1990, pg 89.
4) Deming, Out of the Crisis, pg 61.

10. Eliminate slogans, exhortations, and targets for the work force.

Posters and slogans have long been used in the security arena to raise the level of security awareness. This point refers to those posters or slogans designed to exhort individuals to unreasonable performances when the process or system does not make this possible. For example, don't develop posters that call for no security incidents if the policies, regulations, and tools make it nearly impossible to escape without some "incident". Posters that are designed to remind everybody of the security threat and are aimed at raising security awareness are acceptable as long as they don't do so through increased fear. In addition, any posters should be based not on regulations, but on data collected as part of the security process.

11. Eliminate numerical quotas for the work force.

It is important to remember that establishing quotas for the sake of some reporting purpose is not a good practice. If the goal is set to perform one security inspection a month that is exactly what will be performed whether only one every other month or even one every week is really needed. Setting quotas may also be unreasonable if the tools are not provided to obtain those quotas.

12. Remove barriers that rob people of pride of workmanship.

This is really a problem of communication and tools. If supervisors listen to the feedback of those who work for them, and act on this feedback, the organization runs much smoother. If comments are ignored it raises an invisible, but very real, barrier between the various levels in the organization. In addition, if comments are ignored on how the process can be improved, the individual who saw a way to improve the product or service loses any pride they may have in their work since the message sent to them is it doesn't matter what the quality of the product or service is.

13. Encourage education and self-improvement for everyone.

Organizations need not only good people, they need people that continue to improve through education. Nowhere is this more important than in the computer industry and computer security. We are involved in a business which crosses many disciplines and with the continual and rapid advancements in technology it takes a concerted effort on the part of everybody to stay abreast of developments. To expect this of employees without providing a means to accomplish it and without encouraging it is not only unfair to the employees it is also doomed to fail. If a quality product needs continual adjustments to improve, why shouldn't the same apply to quality people?

14. Take action to accomplish the transformation.

In order for this philosophy to work, enough people in the organization must understand the 14 points to champion the cause. Eventually everyone will have to understand the philosophy and, as has been indicated, everyone will have to be committed to it. Often this new way of doing business is a radical departure from what is being done at the present and it is not without obstacles to its implementation. Top management has to commit to it and then be willing to wait out its possibly slow growth in the organization. As people come to realize its benefits, it will pick up speed and take on a life of its own.

COMPUTER SECURITY AND SPC

While it may at first seem that the terms Statistical Process Control and Total Quality Management are synonymous, this is not the case. They are, however, inseparably linked. TQM really refers to the management commitment to quality and is what the 14 points address. SPC is the tool used to measure the quality of the product or service at various points throughout the process. SPC requires that a number of statistical points of measure be identified in order to monitor the process under observation. A complete list of the statistical points of measure that can be used in computer security are beyond the scope of this paper but a couple of examples may serve to illustrate the relationship between SPC and TQM.

Several tests have been run by organizations to determine how often a system is being "attacked" by unauthorized individuals. What has been extremely interesting in these tests is to see how many of the attacks went unnoticed by the system managers. After a number of these studies have been performed, a statistical average for the number of expected attacks per month can be determined for various organizations (government industry, education ...) and various types of systems (UNIX, VMS ...). If a particular site reports a figure far below the expected value this may be an indicator of a problem and would bear further research. Is, for example, this number low because there actually aren't any attacks or because the site doesn't have the trained personnel or tools necessary to detect the attacks?

Another example of a measurable indicator that we have used in the Air Force relates to "unauthorized" software on personal computers. We all know the threats posed by viruses and how easily they can spread. Consequently in the Air Force no software is to be used until it has been checked out and authorized to be installed. Despite regulations that ban the use of unauthorized software, we know that if we visit any large site we will find systems with unauthorized software. The mere presence of such software, however, is not the issue that it once was. What is significant is when a site's numbers are far from the statistical norm. Too many systems with such software generally indicates a poor security posture and a poor security education program. A number far below what was expected may indicate a good security posture and education program.

The relationship between SPC and TQM can be seen in how the above examples are handled. In the past, a report would be written giving the site some rating based on certain findings. The presence of unauthorized software would have been reported and a poor rating would have been the result. The supervisor for the site might then have issued some statement restating the organizations ban on unauthorized software and directing everyone to delete such software from their systems. The workers would generally comply but within a few months the same software would again start to appear. Instead, if TQM methods were implemented, the report would state that more systems than expected were found to have unauthorized software and that this has been found in the past to be an indication of problems in the security education of the users. As can be seen, in the past we were too often involved with treating the symptoms; now, with TQM, we are interested in finding symptoms which are indicators of problems elsewhere that can be fixed.

A final note on the statistical points of measure is needed. These points of measure are a crucial component in the monitoring of the security process but they must be used in context with TQM and not misused. Most people want to know what these points of measure are so they can immediately start measuring their security posture but care must be taken. Without an initial commitment to TQM any points of

measure cannot be effectively used. The commitment to TQM must come first -- the points will follow.

CONCLUSION

If we accept that we need to commit to TQM, what then do we as security specialists and managers do? The answer is simple to state but harder to implement. Start by adopting the 14 points outlined above. We all can recognize a bit of our own organizations in these points, they are things we have been doing for years. But, as American industry is learning, just because that's the way we have been doing it doesn't mean that its correct or at least can't be improved. TQM is not a quick, overnight fix -- there is none. Improvements will come slowly at first but as more people become convinced of an organization's sincerity in their commitment to quality, the improvements will begin to come faster. This has proven true time and time again in industry.

The hardest part in computer security will still remain that security managers are usually outside the normal line of supervisory authority. We need to demonstrate our commitment to security, not for security's sake but because we recognize that poor security can adversely affect the quality of the organization's product or service. This is the point we need to "sell."

The federal government has begun a push towards quality in its adoption of TQM. President Bush stated in the Federal Total Quality Management Handbook that "The improvement of quality in products and the improvement of quality in service - these are national priorities as never before." The Air Force is committed to this idea and its application in computer security. It is a commitment that we all need to make if we ever hope to wield effective security programs.

REFERENCES

1. Deming, W. Edwards, **Out of the Crisis**, Massachusetts Institute of Technology Center for Advanced Engineering Study, Cambridge, Mass, 1989.

2. Ishikawa, Kaoru, **What Is Total Quality Control? The Japanese Way**, Prentice-Hall, Inc, Englewood Cliffs, NJ, 1985.
3. Juran, J. M. and Frank M. Gryna, Jr., **Quality Planning and Analysis**, McGraw-Hill Book Company, New York, NY, 1980.

4. Scherkenbach, William W., **The Deming Route To Quality and Productivity**, CEEPress Books, Washington D. C, 1990.

5. Walton, Mary, **The Deming Management Method**, Perigee Books, New York, NY, 1986.

CONCEPT FOR A SMART CARD KERBEROS

Marjan Krajewski, Jr.
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730-1420

Abstract

This paper addresses security issues associated with authenticating users to system services in distributed information systems. Its focus is the presentation of a concept for augmenting the Kerberos distributed system identification and authentication protocol through the integration of emerging smart card technology. The goal is to protect against the threat from malicious workstation-resident Trojan Horse programs capturing a user's authentication data for later use by an intruder and other, related security problems.

Keywords: authentication, unitary login, Kerberos, smart cards, distributed systems, network security

Introduction

Two critical aspects of information system security are the application of access controls based on a user's authorizations and the creation of an audit trail based on a user's actions [1]. Both are dependent upon the accurate authentication of users to guard against the threat of intruders masquerading as valid users. Traditionally, a user is authenticated to a host upon presentation of a valid combination of userid and password. In a distributed processing environment, a user often needs to access resources located at multiple servers from multiple workstations interconnected via a communications network. Authentication to each host accessed is crucial, but presenting separate userid/password pairs can be both unwieldy and unsecure. What is needed is a mechanism which requires users to identify and authenticate themselves once to a trusted agent which then performs the necessary user identification and authentication to each accessed resource transparently (unitary login).

Background

Previous work in developing secure unitary login protocols for distributed systems include those intended for open environments (e.g., the Massachusetts Institute of Technology Kerberos protocol [2], the Carnegie Mellon University Strongbox protocol [3], and the ISO OSI Directory Services protocols [4]) and those intended for closed environments (e.g., the World Wide Military Command and Control System (WWMCCS) Information System Network Authentication Service (WISNAS) protocol [5], and the Department of Defense Intelligence Information System (DoDIIS) Network Security for Information Exchange (DNSIX) protocol [6]).

Each of these protocols provides different authentication services (e.g., Kerberos, WISNAS, and DNSIX are more connection-oriented while Strongbox and the OSI Directory Services are more process-oriented) and depends upon different mechanisms to provide security (e.g., Kerberos employs conventional encryption, Strongbox employs "zero-disclosure" proofs, OSI Directory Services employs public key encryption). None of them are intended for a truly hostile environment (i.e., one subject to active attacks against both workstations/servers and the network). WISNAS and DNSIX, though designed for military applications, do not use any form of encryption and, as such, are intended for physically secure environments with trusted users and no eavesdropping threats. With these protocols, any one of a number of commercially available

network protocol analyzers can easily intercept sensitive authentication information if allowed physical access to the network. The other protocols protect against the threat of network eavesdropping but assume that workstations and servers are protected by other mechanisms (e.g., physical ownership/control). The covert introduction of a Trojan Horse program into these workstations can easily "break" the authentication mechanism. Both Government and non-Government organizations could greatly ease the problems associated with password management and the threat from masquerading on their increasingly distributed information systems with a unitary login capability which was secure from both a workstation/server and a network perspective.

The Kerberos protocol possesses many advantages as a basis for this capability. Originally developed to provide user authentication for the distributed open computing environment of MIT's Project Athena, Kerberos is growing significantly in popularity (it has been adopted by the Open Software Foundation and Unix International as well as being offered in several commercial products). It uses algorithm-independent conventional (private) key encryption to protect against network eavesdropping. This latter feature is especially important for military/intelligence applications in that the current Data Encryption Standard (DES) algorithm might be inadequate for certain environments. If so, it can easily be replaced with a stronger algorithm.

Kerberos Overview

Begun in 1983, Project Athena is MIT's investigation of advanced computer technology in the university curriculum. Kerberos was developed under Project Athena as an authentication system that can be added to existing distributed computing environments with minimal modification of existing applications.

Kerberos utilizes a trusted central authentication server, referred to herein as the Kerberos Authentication Server (KAS). This central server contains a database of system entities (registered users and services) and their private cryptographic keys. These private keys, known only to the respective entity and the KAS, allow the KAS to communicate privately with the Kerberos agent of each system service (referred to herein as server Kerberos) and with the Kerberos agent of each registered user who wishes to be logged in (referred to herein as client Kerberos). The central server also contains a ticket granting service to provide a trusted means for logged in users to prove their identity to system services. Finally, it contains a key generation service which supplies authorized pairs of entities with temporary cryptographic keys (session keys).

The Kerberos protocol is based on the concept of *tickets* and *authenticators*. A ticket is issued by the KAS for a single user and a specified service. It contains the serviceid, the userid, the user's (workstation) address, a timestamp, the ticket's lifetime and a randomly chosen session key to be used by this user and this service. This information is protected by encryption under the service's private key. Since this key is known only to the service and the KAS, the service is assured of the authenticity of the ticket. Once a ticket is issued, it can be used many times by the named user to gain access to the indicated service until the ticket expires.

Unlike the ticket, the authenticator is built by client Kerberos. A new one must be generated every time the user wishes to use a ticket. An authenticator contains the user's id, the user's (workstation) address, and a timestamp. The authenticator is encrypted with the session key which is associated with the ticket. Encryption of the authenticator provides integrity of the authenticator and assures the service that the user is the system entity who received the original ticket. The further agreement of the user id in the authenticator with the one in the ticket and the address with the one from which the ticket arrived provides further assurance. Agreement of the timestamp with the current time assures the service that this is a fresh ticket/authenticator pair and not a replay of an old pair.

A new user or the administrator of a new system service must first register with Kerberos. Registration consists of making a new entry in the KAS database and issuing an id and private key. In the case of a user, the private key is issued in the form of a password. The administrator of the system service must load the service's private key into the server Kerberos software. Following registration, user interaction with Kerberos consists of three phases. The first phase occurs during login (Figure 1).

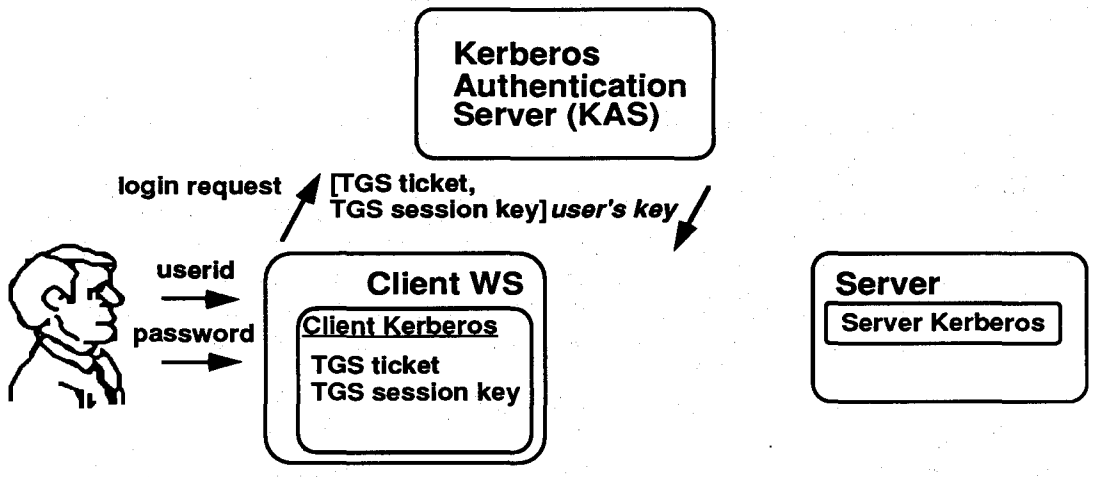


Figure 1. Kerberos Authentication - User Login

The user first enters his userid into the workstation and sends a request to the KAS. A ticket to the ticket granting service and its associated session key are then generated by the KAS and sent to the user in a message encrypted in a private key derived from the user's password. The user then enters a password and, if correct, the message is decrypted and the ticket granting service ticket and session key are obtained. If the entered password is incorrect, the message will not be decrypted. In the second phase (Figure 2), the user, desiring to access a specific system service, presents the ticket granting service ticket and an associated authenticator to the ticket granting service to request a ticket for the desired system service.

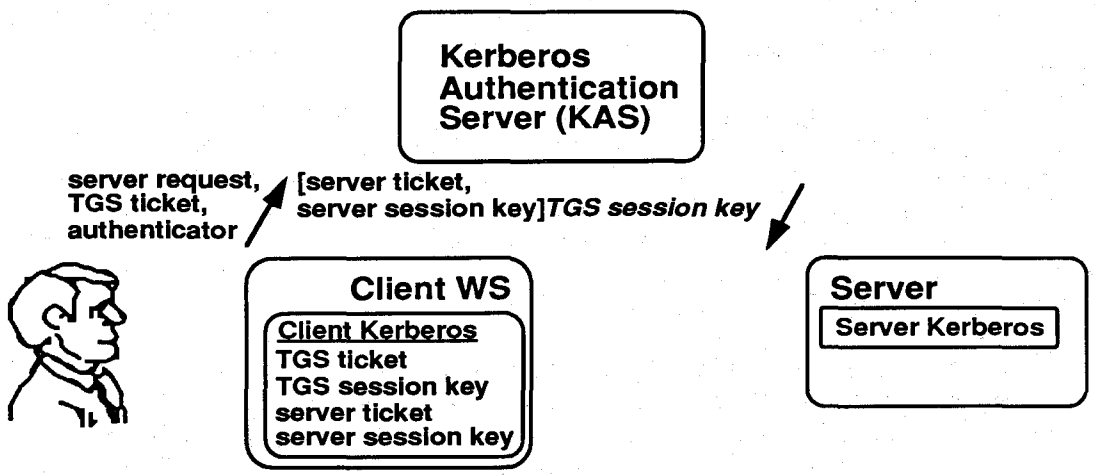


Figure 2. Kerberos Authentication - Obtaining a Server Ticket

The ticket and associated authenticator identifies and authenticates the user to the ticket granting service. The ticket granting service then generates and sends an appropriate server ticket and session key to the user, encrypted in the session key associated with the ticket granting service.

This message is then decrypted within the user's workstation. In the third phase (Figure 3), the user generates an appropriate authenticator for the desired server ticket, presents the server ticket and associated authenticator to the service, and, following validation by the server, obtains access.

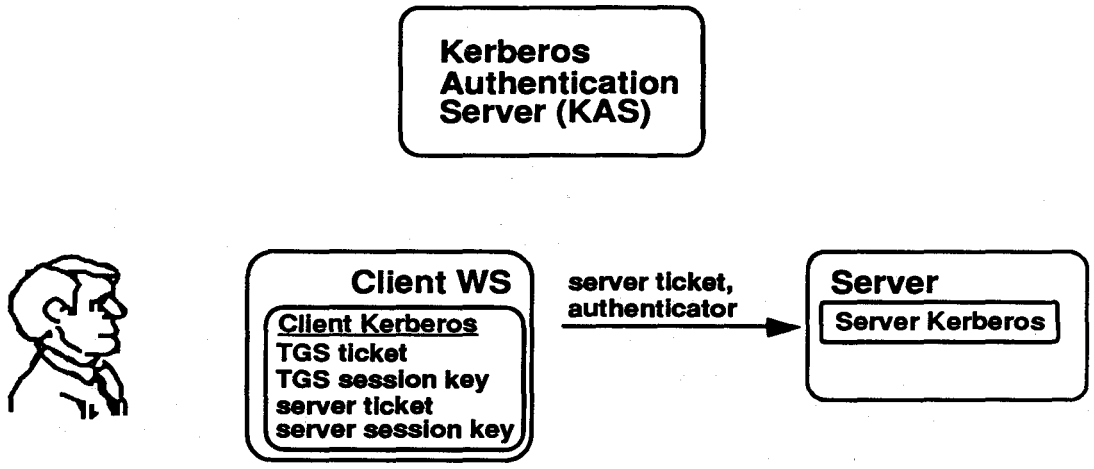


Figure 3. Kerberos Authentication - Accessing a System Service

Kerberos Security Issues

Kerberos has been analyzed from a general security perspective [7]. A significant vulnerability involves its manipulation of the user's Kerberos password and other sensitive authentication information (i.e., session keys) within the workstation, thereby making it vulnerable to Trojan Horse threats which could capture such data for later use by an intruder[1]. Another vulnerability involves the threat of repeated attacks at the intruder's leisure following interception of the initial message from the central authentication server. This message contains the ticket granting service ticket and associated session key and is encrypted by a relatively weak password-derived key. A third vulnerability involves the inherent weakness of depending solely upon a single factor (i.e., a password) for the initial user authentication. Passwords can be easily borrowed or stolen. These vulnerabilities are depicted in Figure 4.

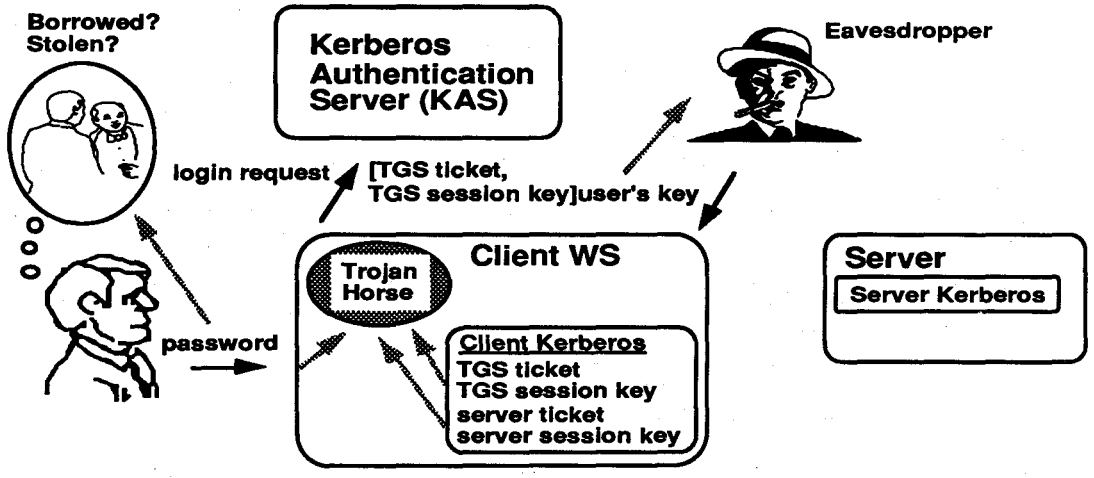


Figure 4. Kerberos Login Vulnerabilities

[1]This particular vulnerability is not unique to Kerberos. It represents a known vulnerability of any authentication mechanism which relies upon the user entering sensitive data into the workstation.

Kerberos Augmentation Concept

Advances in encryption and smart card technology have reached the point where significant amounts of information can be stored and significant processing can be performed entirely within the isolated environment of the card itself. When this technology is combined with user-unique information (e.g., a password) that is useless except when processed by the appropriate smart card, a significantly stronger authentication mechanism can be constructed than is available with "standard" (i.e., software-only) Kerberos.

The concept described here augments Kerberos security by *moving all cryptographic processing from the workstation into a user-unique smart card and storing the user's private key in encrypted form on the smart card* (Figure 5). The user's private key would be encrypted *in a key derived from a password.* In this way, neither possession of the card alone nor knowledge of the password alone would be sufficient to authenticate a user. Encryption and decryption operations and the storage of unencrypted authentication information would occur only within a trusted processing environment (i.e., that of the smart card).

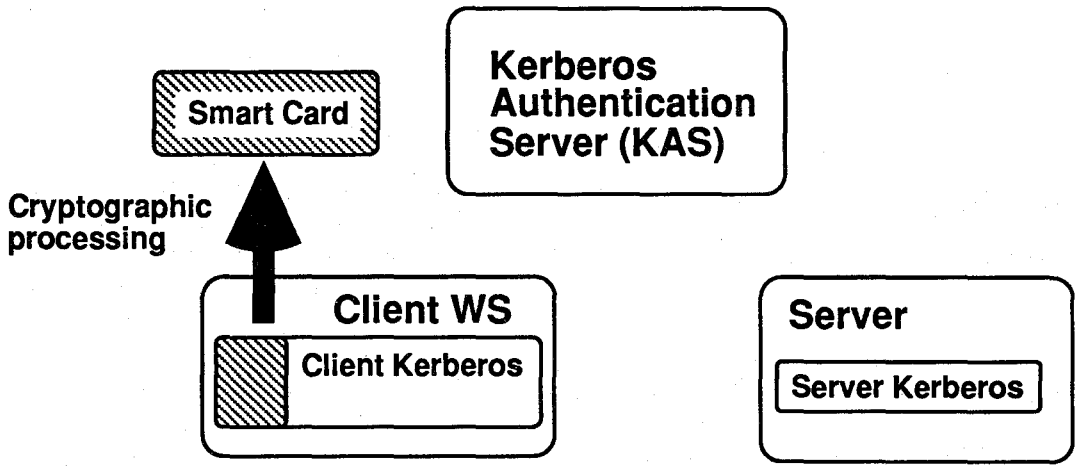


Figure 5. Kerberos Augmentation Concept

In proposing Kerberos augmentations, an important constraint involves maintaining interoperability with existing Kerberos implementations. Observing this constraint allows augmented Kerberos components to be gradually introduced into an operational environment as time and resources permit. This constraint mandates that neither the KAS nor the server Kerberos implementations be affected in any way. The concept presented here limits all modifications to the client Kerberos software residing in the user's workstation. It is depicted in Figure 6 and described in the following text.

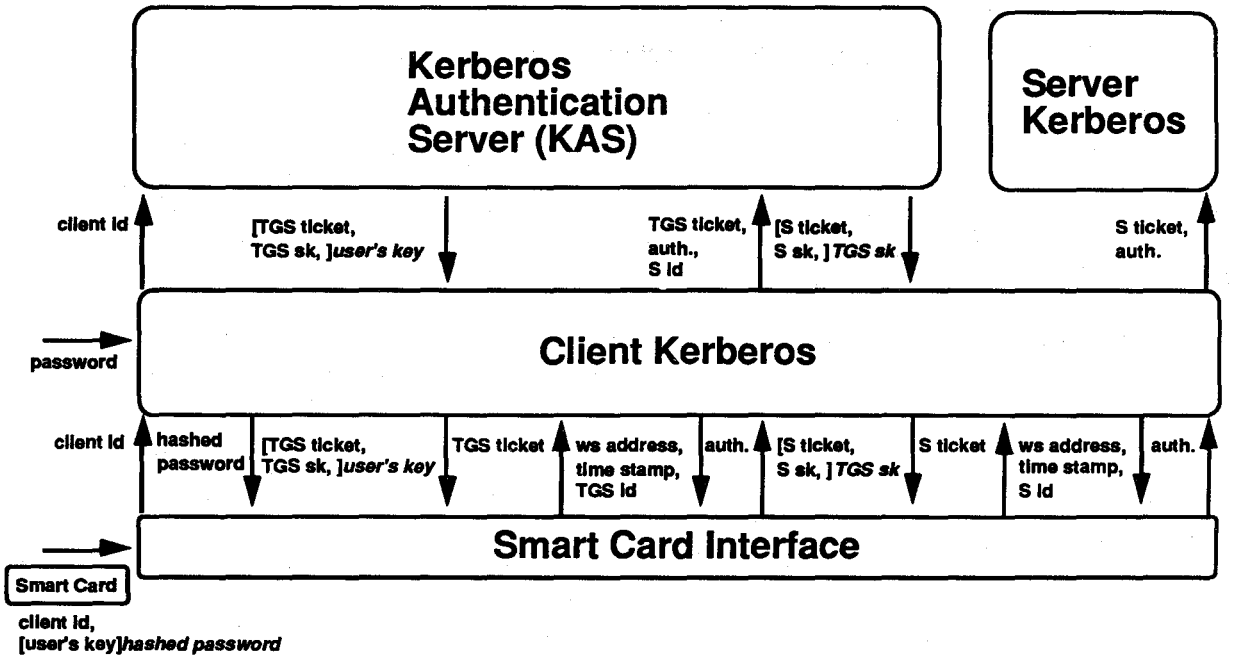


Figure 6. Smart Card Augmented Kerberos

The smart card augmented Kerberos would function as follows:

Initial State: Client Kerberos holds no user-unique information. The smart card holds the userid in the clear and the user's private key encrypted in a key derived from a password.

Step 1: The user inserts his/her smart card into the card reader attached to the workstation. Client Kerberos commands the smart card to transfer the userid.

Step 2: Client Kerberos sends the userid to the KAS and prompts the user for a password. Client Kerberos derives a key from the password.

Step 3: The KAS generates the user's ticket granting service (TGS) ticket and associated TGS session key, encrypts them in the user's private key, and sends the encrypted message to client Kerberos.

Step 4: Client Kerberos transfers the key derived from the user's password and the encrypted message from the KAS to the smart card.

Step 5: The smart card uses the key derived from the user's password to decrypt and obtain the user's private key. The smart card then uses the user's private key to decrypt the encrypted message from the KAS and obtain the TGS ticket and the associated TGS session key. The smart card stores the TGS session key in its volatile memory, destroys the key derived from the user's password and the decrypted copy of the user's private key, and transfers the TGS ticket back to client Kerberos.

Step 6: To access a system service, client Kerberos first determines if a ticket to that service is needed (one may have been obtained earlier and, if so, the process jumps to step 12). If a ticket is needed, client Kerberos transfers the workstation address, a timestamp, and a request for a TGS authenticator to the smart card.

Step 7: The smart card then creates an authenticator for the TGS service by encrypting the userid, workstation address, and timestamp in the TGS session key. It then transfers the authenticator to client Kerberos.

Step 8: Client Kerberos sends the service request, together with the TGS ticket and authenticator, to the KAS.

Step 9: The KAS generates the appropriate server ticket and associated server session key, encrypts them in the user's TGS session key and sends the encrypted message to client Kerberos.

Step 10: Client Kerberos transfers the encrypted message to the smart card.

Step 11: The smart card decrypts the message from the KAS using the TGS session key to obtain the server ticket and the associated server session key. The smart card stores the server session key and transfers the server ticket to client Kerberos.

Step 12: Client Kerberos transfers the workstation address, a timestamp, and a request for a server authenticator to the smart card.

Step 13: The smart card creates an authenticator for the requested server by encrypting the userid, workstation address, and timestamp in the server session key. It then transfers the authenticator to client Kerberos.

Step 14: Client Kerberos sends the server ticket and authenticator to the requested service.

Step 15: Server Kerberos decrypts the server ticket and authenticator and makes an access control decision.

Conclusions

The use of a smart card in the manner described above improves system security in three significant ways. It requires a user to provide both something he/she possesses (i.e., a smart card) as well as something he/she knows (i.e., a password). Either item alone is useless. This significantly reduces the risk from password borrowing/theft. It allows the initial message from the central authentication server to be encrypted in a truly random key (i.e., the user's private key need not be derived from a password). A cryptographic attack on this message must therefore assume that the entire keyspace is available for use. This significantly reduces the risk from network eavesdropping. Finally, it ensures that only encrypted data is processed by a user's workstation. Software residing on a workstation can view only the same data a network eavesdropper can view (and a password tied to a specific smart card). This significantly reduces the risk from Trojan Horse programs.

Potential issues that might arise in attempting to realize this concept include those related to feasibility, security, and performance. Regarding feasibility, it is not clear how easily current Kerberos implementations and current smart card technology will support the required functional partitioning between inboard and outboard elements. Regarding security, the movement of all sensitive processing into a smart card mandates that the smart card provide a trusted environment which is incorruptible from the workstation. It is not clear whether current smart card technology can provide the needed isolation. Regarding performance, the use of an outboard microprocessor will undoubtedly impact response time, but whether or not the degradation is acceptable to the user is critical. These issues can best be addressed through implementation and evaluation.

Kerberos is evolving to provide more flexibility and greater security [8]. These enhancements facilitate smart card augmentation. Smart card technology is also advancing rapidly and advanced systems are currently under development by numerous vendors with the capability to provide the necessary processing [9]. The ideal smart card candidate will possess on-board general purpose computing, several kilobytes of non-volatile and volatile memory, and anti-tamper features. One example of current technology is the OmegaCard™ by Sota Electronics, Inc., Agoura Hills, California. It contains an Intel 8051 custom microcomputer, 8 kilobytes of non-volatile memory, and a mini operating system. Our current plans are to use this device as the basis for a proof-of-concept demonstration and evaluation.

References

[1] National Computer Security Center, "Department of Defense Trusted Computer System Evaluation Criteria," DoD Standard 5200.28-STD, December 1985.

[2] Steiner, J., Neuman, C., and Schiller, J. "Kerberos: An Authentication Service for Open Network Systems," USENIX Conference, 1988.

[3] Yee, B. S., Tyger, J. D., and Spector, A. Z., "Strongbox: A Self-Securing Protection System for Distributed Programs," Department of Computer Science, Carnegie-Mellon University, Technical Report CMU-CS-87-184, 5 January 1988.

[4] ISO, "International Standard ISO 9594-8: Information Technology - Open Systems Interconnection - The Directory - Part 8: Authentication Framework," 1990.

[5] GTE Government Division, "WIS Network-User Authentication Service (WISNAS)," Strategic Systems Division, WIS-85-TM-16, 15 March 1985.

[6] Defense Intelligence Agency, "DoDIIS Network Security Architecture and DNSIX," DoDIIS System Engineering Office, DRS-2600-5466-86, May 1986.

[7] Bellovin, S. M. and Merritt, M., "Limitations of the Kerberos Authentication System," Computer Communications Review, October 1990.

[8] Kohl, John T., "The Evolution of the Kerberos Authentication Service," Spring EurOpen Conference, 1991.

[9] Bright, R., Smart Cards: Principles, Practice, Applications, Ellis Horwood Limited, distributed in the US by Halsted Press, a division of John Wiley & Sons, 1988.

CONCEPT PAPER --
AN OVERVIEW OF THE PROPOSED
TRUST TECHNOLOGY ASSESSMENT PROGRAM

Ellen E. Flahavin
Patricia R. Toth

Computer Security Division
National Institute of Standards and Technology

ABSTRACT

This paper provides an overview of the philosophy, objectives, and methodology of a proposed new program for the evaluation of trusted information technology products. The program will focus on products with the features and assurances characterized by the TCSEC (Trusted Computer System Evaluation Criteria, or Orange Book) B1 and lower levels. The program is product, not application, oriented - no attempt can be made to separate products intended to process classified information from those intended for other applications. The program is intended to be fully compatible with the Federal Criteria Version I. The program will continue to emphasize the credibility and fairness of the evaluation process. The program will allow a seamless transition from the current process in which NSA alone evaluates products and populates the EPL. The new program is to be called the Trust Technology Assessment Program (TTAP). The program recognizes the managerial and technical benefits of NIST/NSA cooperation, and so incorporates mechanisms to utilize the cooperation, and mechanisms to equitably and efficiently resolve disagreements.

A. INTRODUCTION

1. PHILOSOPHY

The TTAP will establish, approve and oversee a number of Accredited Trust Assessment Laboratories (ATALs) focusing on products with the features and assurances characterized by the Orange Book's B1 and lower levels. The program will use the Federal Criteria as soon as it is available, and will develop mechanisms for other criteria if appropriate.

The TTAP approval and oversight mechanisms will assure continued quality and fairness; they will be modelled on NIST's existing National Voluntary Laboratory Accreditation Program (NVLAP). To support consistent product evaluations at multiple sites, TTAP will develop standardized testing and analysis procedures. These

standardized procedures will be the basis for mutual recognition of evaluation with other nations. (European Community ITSEC/ITSEM evaluations are performed under the purview of national test standardization bodies associated with NVLAP.)

We envision that market forces, now frustrated by severely limited evaluation capacity, will drive costs and evaluation time down once the ATALs are operating. More products, especially current products will be available.

2. HISTORY AND RATIONALE

To date, only NSA performs the trust evaluations which place products on the EPL. Over the last five years , there has been a marked improvement in baseline computer security expertise. Vendors now provide more high trust products; and have become skilled and quite prolific with B1 and below products. The good computer practices which make a product "trusted" also make it bug-free, maintainable, and up-gradeable. The same improvement in computer security expertise among vendors has also occurred among a variety of Government organizations and contractors, as well as a community of security consultants.

The challenges of network and information system security are providing NSA with more work at higher levels of trust. Technology is providing users more products with claimed security features. A growing body of customers - users and system security accreditors especially - need to have a fair, impartial expert evaluation of the numerous, often confusing vendor security claims. And technology has made available a growing body of expertise which can be tapped to support the evaluation of those products.

It remains essential that a wide variety of current technology trusted products at the lower levels of assurance (B1 and below) be readily available to US Government users. These products must be up-to-date, affordable and adaptable to a wide variety of user needs and applications. Vendors confronting a speedy, widely available evaluation capability will have market incentives to be better prepared to enter evaluation, to maintain evaluated ratings, and so to meet the need for those products.

3. GOALS AND OBJECTIVES

A key TTAP objective is to achieve a greater number of evaluated products - products available as Commercial Off The Shelf (COTS) solutions for the non-classified as well as the classified community - without sacrificing the level of quality of the individual product evaluations.

A second major goal is to provide the US basis for mutual recognition of product evaluations with the community using the ITSEC.

The TTAP will minimize the time and cost of evaluations, and will maximize product availability, all consistent with the level of assurance required and without loss of quality.

The TTAP will develop methods for evaluation in accordance with the Federal Criteria, and aligned as appropriate with the European Community's ITSEC and ITSEM. This will assist US-based vendors to have a single, widely accepted evaluation. This harmonization of evaluations will be complemented by efforts to harmonize the criteria.

Evaluation of products is always important to those who integrate products into systems. It is the goal of the TTAP to provide more evaluations, and more evaluation reports, which will support the system integrators and accreditors who use evaluated products.

Specific objectives of the TTAP are to:

a. Provide users, especially Federal government users, with evaluation of the products they require, when they are required.

b. Provide a single evaluation which will allow worldwide product acceptance for US-based vendors.

c. Provide a basis for system integrators, users, and accreditors to understand product security functionality, and to have confidence that the features and assurances have been evaluated.

d. Save the government and the vendor time, effort and cost in the evaluation process.

e. Use the NSA expertise in trust wisely - maintain its critical mass, apply it to most important national security relevant products, and use it to teach others.

B. OVERVIEW OF THE METHODOLOGY

1. PRODUCT EVALUATION MANAGEMENT BOARD

The Directors of NSA and NIST will appoint a Product Evaluation Management Board (PEMB), with policy and operational oversight of the TTAP. The PEMB will be responsible for overall quality of evaluations; and for overseeing mutual recognition arrangements with other nations. The PEMB will also be responsible for all operational aspects of the TTAP, including conduct of the Technical Review Boards, grant of B1 and below product trust ratings and oversight of the ATALs.

a. Only the PEMB will have the authority to grant a B1 and below product trust rating. The rating will be granted only if both NSA and NIST agree.

b. Ratings granted by the PEMB are intended to be recognized nationally and internationally.

c. The PEMB will be responsible for maintaining liaison with the product evaluation authorities of other nations and assuring satisfaction of ongoing mutual recognition arrangements.

d. The PEMB will be responsible for administering the procedures of the NVLAP for ATALs established by that program. This includes laboratory facility and procedural quality control, oversight of the ATALs, and sponsorship of evaluator technical training.

e. The PEMB will be responsible for general advice and support of the ATALs, including provision of official interpretations of criteria as required.

3. EVALUATED PRODUCTS LIST

There will continue to be a single US Government Evaluated Products List (EPL). This EPL will be populated with products evaluated under all of the approved processes described below. That portion of the EPL which covers products evaluated under the TTAP will be administered by the PEMB.

4. PRODUCT EVALUATION LABORATORIES

Four types of evaluation labs are envisioned to perform product evaluations in compliance with the TTAP and the new Federal Criteria. These include NSA's existing Trusted Product and Network Security Evaluations Division (NSA), operating independently according to its own requirements, plus three basic types of Accredited Trust Assessment Laboratories (ATALs). All three types of ATALs will be established and overseen via the NVLAP process, and may be operated by product vendors, commercial organizations, or government agencies. All ATALs will be required to meet rigorous facilities, personnel, procedural and oversight requirements specified by NVLAP. ATALs will be approved to perform specific standardized tests and analytic procedures for evaluation of products against the FC in the range of trust described by current C2-B1 ratings.

a. First-Party ATALs

First-party ATALs are private testing labs operated by computer product vendors for evaluation of their own products. These ATALs will be limited initially to RAMP and PORT evaluations

only. Upon final approval of this evaluation method, first party ATALS must meet the following requirements:

1. First-party ATALs will be required to demonstrate strong independence from the parent firm, with full opportunity to conduct stringent product evaluations.

2. First-party ATALs would be accredited by NVLAP to evaluate their company's products for conformance to the FC via performance of standardized tests and analytic procedures.

3. They would submit evaluation reports to the PEMB for acceptance.

4. Oversight could include PEMB auditing of the evaluations under specified conditions.

b. Third-Party Commercial ATALs

Third-party "commercial" ATALs are private and independent testing labs, normally operated for profit and intended to accept any products for evaluation that sponsors are willing to pay for.

1. Third-party ATALs would be accredited by NVLAP after approval by the PEMB to evaluate any sponsor's products for conformance to the FC via performance of standardized tests and analytic procedures.

2. They would submit evaluation reports to the PEMB for acceptance.

3. Evaluation sponsors will typically include product vendors or major Federal agency or commercial product users.

4. Degree of third-party lab independence from commercial sponsors, management of proprietary information, and potential for conflict of interest are important considerations.

5. Oversight could include PEMB auditing of the evaluations under specified conditions.

c. Third-Party Government ATALs

Third-party Government ATALs are testing labs established within Federal agencies (such as military Services), which will evaluate any products that agency sponsors desire.

1. Products of interest would typically include embedded or complex systems developed specifically to meet the agency's mission or products of agency interest not being evaluated by other types of labs.

2. Third-party ATALs would be accredited by NVLAP after approval by the PEMB to evaluate products for conformance to the FC via performance of standardized tests and analytic procedures.

3. The agency could submit evaluation reports to the PEMB if the agency desires the evaluations to be recognized nationally and EPL ratings granted.

d. NSA and High Assurance Evaluations

NSA will be considered a fully-independent special-case third-party Government ATAL for TTAP purposes, without PEMB or NVLAP compliance requirements.

1. NSA will continue to be the sole organization to evaluate products at levels of trust currently described by B2 or higher.

2. NSA will continue to operate according to its own requirements as it does currently.

3. NSA will maintain appropriate internally-specified processes and levels of rigor, using any methods and criteria (including the FC and TTAP if desired) deemed appropriate to meet its customers' needs.

4. In addition, NSA will have the option of evaluating any products of any type needed to meet its customer requirements.

5. TECHNICAL REVIEW BOARD

After any testing lab completes an evaluation of a product, the lab will forward a formal evaluation report to the PEMB. The PEMB will convene the Technical Review Board (TRB) to review all aspects of the product evaluation. The TRB will be an independent organization acting under the guidance of the PEMB and consisting of employees from the organizations making up the PEMB and their contractors. The TRB will conduct a technical review of evaluation reports, investigate the evaluation methods used, question the evaluators for consistency between evaluations, and recommend an evaluation rating or refer the evaluation back to the testing lab. Upon completion of its actions, the TRB will forward its recommendations to the PEMB.

6. VENDOR ASSISTANCE

Vendors with potentially evaluatable products could obtain assistance and gain initial entry into the TTAP process as follows:

a. Vendor applies to the PEMB for entry into the process. The PEMB will assist the vendor to identify the appropriate type of testing lab for the product. If appropriate, the vendor will be given guidance and initial assistance in establishing a first-party product testing lab for NVLAP accreditation.

b. Independent testing labs may provide assistance to vendors on a consultative basis to prepare products for evaluation.

C. STEPS TO ESTABLISH TTAP

In order to establish this program, the following steps are required (exact order of the steps must still be determined and further refinements developed):

1. Initiate TTAP Working Group

It is essential that a joint NIST-NSA Working Group be set up without delay to do detailed planning and coordinate implementation of the TTAP. This Working Group should be jointly led by NSA and NIST and staffed with a small core of selected employees and senior contractors.

2. Establish Prototype Product Evaluation Management Board

Organizational structure and initial rules for the PEMB, defining its role, membership and general operating procedures, and procedures for granting/revoking ratings will be established, will be established, and the PEMB will be brought into operation. The prototype PEMB will initially consist of the Chiefs of NSA's Office of INFOSEC Developmental Systems Security Evaluations and NIST's Computer Security Division.

3. Agree Upon Federal Criteria and Approach

The Federal Criteria, including both functionality and assurance requirements, will be agreed upon by NIST and NSA.

4. Establish Evaluator Qualifications

Product evaluator professional qualifications, including initial education requirements, specific evaluator training, and experience requirements, will be identified. These qualifications will be established at a minimum of three levels: initial entry, full team member, and team leader.

5. Establish NVLAP Requirements for ATALs

Initial NVLAP accreditation requirements for first and third party testing labs will also be established and tested. These requirements will include facilities, management, personnel, and testing administration.

6. Establish Prototype Tests and Procedures

Prototypes of standardized analytic procedures and tests will be initially developed and then evolved for the low end functionality and assurance requirements.

7. Initiate Pilot Test

A pilot test of the low-end evaluation process will be conducted by NIST and NSA.

8. Develop Evaluation Guidance

Once pilot testing and analytic procedures are working well, formal guidance (handbooks/manuals) on conduct of evaluations for vendors, labs, and lab personnel will be established.

9. Establish Initial ATAL Sites Under NVLAP

Using previously-developed NVLAP accreditation guidance, establish one or more volunteer candidate labs as ATALs. Expand NVLAP accreditation procedures to accommodate lessons learned.

10. Establish Evaluator Training Program

Using training requirements determined previously, build modules of instruction to meet required topics, identify qualified instructors and training facilities.

11. Establish Rules for TRB

Rules for the new Technical Review Board, defining its role, membership, and procedures for reviewing evaluations and making recommendations to the PEMB will be established.

12. Prepare Transition Plan for Products in Current System

Transition plan for incorporating previously-evaluated products and products currently under evaluation into the new program will be developed.

13. Develop Ratings Maintenance Guidance

Guidance to vendors for maintaining product ratings under the TTAP will be developed.

14. Develop Product Ratings Usage Guidance

Guidance for users and procurement personnel on how to use the product ratings will be developed.

15. Develop Vendor Guidance on Building Trusted Products

Guidance for vendors on how to design and build trusted products to these new requirements will be developed.

D. CONCLUSION

This proposed TTAP program could provide a win-win situation. The vendor can win because quality trusted product evaluations could be completed more quickly with less expense and effort. The government and private sector user communities can win because trusted products could be brought to the marketplace more quickly and the costs for a trusted product may become lower as the vendors pass their savings on to the purchasers. The user communities and vendors all could win as an increased number of current-technology trusted products becomes available. The vendor could sell more products and the users could have the opportunity to purchase a wider variety of products. The vendor could also be able to broaden their market base through mutual recognition of product ratings with other countries.

CURRENT ENDORSED TOOLS LIST (ETL) EXAMPLES RESEARCH LESSONS LEARNED

Cristi Garvey, Aaron Goldstein, Eric Anderson

TRW Systems Integration Group
Redondo Beach, CA 90278

Abstract

The Current Endorsed Tools List Examples (ETL) Project[1] developed a simple, yet realistic, worked example of A1 verification technology. This example was developed on schedule and within budget by a novice verifier. The project produced many lessons learned. The most important of these to the research community are the following:

- Prototyping the design verification process reduces rework of both the system design and of the verification tasks.
- Using the Deductive Theory Manager and Gypsy Reprover tools make the Gypsy Verification Environment user more productive.
- By applying a combination of techniques, the Gypsy Information Flow Tool can be used on a significant example.

Introduction

The Current ETL Examples project addresses a need for publicly available worked examples demonstrating the use of design verification technology in system development. At the time this project began, few examples of successful A1 verification efforts had been completed; even fewer were available in the public domain. Furthermore, those very few examples that were available were either too complex to be understood by novice users of verification technology or too simple to be of any help in verifying real systems. This has resulted, understandably, in a general lack of enthusiasm for the use of verification technology in the development of secure systems. System builders are not encouraged to use formal verification techniques, since they have not seen the benefits of the significant effort that is required to apply them. Likewise, they are not encouraged to use automated verification tools, since the available tools have not been sufficiently field tested. Worse yet, the limited application of such tools has amplified the problem by denying the tool developers sufficient opportunities to field test their products.

The goals of this project were: 1) to develop a worked example of the application of verification technology; 2) to assess the effort required and the benefits obtained in doing so; and 3) in the process, to develop new sources of verification expertise.

These goals were realized in a small-scale example of design verification for a trusted computer system. The example involved verification of a simple, hypothetical, small computer system that was designed to meet requirements at Class A1 of the *Trusted Computer System Evaluation Criteria* (TCSEC)[Cen85]. That computer system was modeled, specified, verified and analyzed using state of the art verification tools and techniques. All of the verification tasks were performed by a software engineer who had no prior experience using verification technology, but who had significant experience in developing software for secure systems. The verifier was trained on the job by a verification consultant (an expert in the use of formal verification tools and techniques). In addition to training the verifier and reviewing his work, the verification consultant was responsible for installing and maintaining the automated verification tools. The verifier's experiences (both good and bad) in applying verification technology to the design of the example system were carefully documented[TRW91a]. In addition, labor (i.e., man-hour) expenditures for each verification activity were precisely recorded.

System Description

The example system is referred to as the Kernel File Manager (KFM). This hypothetical system is as-

[1]This project was performed under contract number MDA904-90-C-7058 for the Department of Defense.

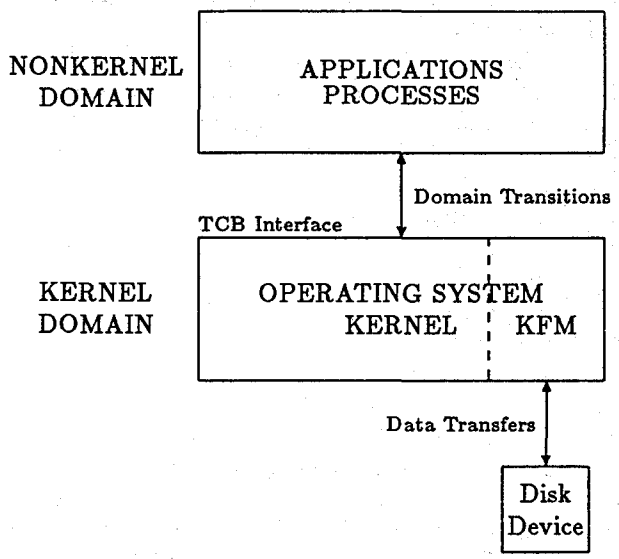


Figure 1: KFM Operational Environment

sumed to serve as a subsystem of a multilevel secure operating system. More specifically, it is assumed to be a component of the operating system's security kernel. Its purpose is to manage all disk resources under the control of the operating system. Figure 1 illustrates this relationship between the KFM and its operational environment.

The KFM organizes the storage on a disk device into logical entities called *files*. Processes do not access the disk device directly; they may only access files. Furthermore, they may do so only through a limited set of operations provided by the KFM. These operations, termed *kernel services*, are invoked by means of a secure domain transition mechanism that is assumed to be implemented by the operating system. The KFM provides kernel services for creating and deleting files, opening and closing them, reading and writing their contents, and modifying their discretionary access attributes.

The security policy for the KFM[TRW91] is derived from security policy requirements at Class A1 of the TCSEC, which are in turn derived from United States Department of Defense (DoD) policies such as those described in *DoD Directive 5200.28*[DoD88]. The KFM security policy includes the mandatory and discretionary access policies outlined in the TCSEC, which are primarily concerned with disclosure of information. It does not include policies concerning data integrity, however.

The kernel services provided by the KFM are typical of the kinds of services provided by real operating system security kernels. They include services that create and delete objects, services that initiate and terminate access to existing objects by existing subjects, services that transfer information from subjects to objects and vice versa, and services that modify discretionary access permissions for objects. Specifically, the KFM controls access between processes (subjects) and files (objects). Two modes of access are supported: Read (observation only) and Write (observation and alteration). Mandatory access control decisions are based on a comparison of the security levels, each consisting of a hierarchical classification and a non-hierarchical set of categories, associated with each subject and each object. Discretionary access control decisions are based on a comparison of the subject's desired mode of access against the allowed modes of access indicated by the user access control list and group access control list associated with each object.

Although the KFM is actually part of a larger system, it is treated as a separate system for the purposes of this example. Verification of a complete operating system security kernel is too large a task for a tutorial example such as this. Limiting the functionality of the example system to a small set of services typical of those provided by real operating system security kernels produces a smaller, more useful, example of the application of verification technology.

Design Process

The design process for a Class A1 system, such as the KFM, involves both traditional design activities and formal verification activities. The former include designing the software to meet its requirements and writing an informal specification of the top level design (i.e., a Descriptive Top Level Specification (DTLS)). The latter include producing a formal model of the security policy, informally demonstrating that the DTLS is consistent with the security model, writing a Formal Top-Level Specification (FTLS) of the design, mathematically proving that the FTLS is consistent with the security model and performing a covert channel analysis.

In designing the KFM, we decided to prototype the design process for a single kernel service prior to designing and verifying the rest of the system. There were two reasons for this decision. First of all, we needed a vehicle for rapidly training the inexperienced verifier. In taking a single kernel service through the complete design process, the novice verifier would rapidly become acquainted with all of the tools and techniques necessary for verification of the remaining kernel services. Secondly, we needed to re-

fine our specification style and verification approach. By doing this early in the design process, we hoped to avoid costly rework.

Our specification style and verification approach were based on those developed for verification of the Army Secure Operating System (ASOS) kernel[BLD90][BDCG90]. In addition to incorporating many of the lessons learned from the ASOS verification effort, we revised the specification style and verification approach to allow for the use of new tools – in particular, the Deductive Theory Manager (DTM)[BDCG90] and the Gypsy Reprover[TRW90b] Tool. We also developed a more workable approach for using the Gypsy Information Flow Tool (GIFT)[JM89] to automate covert channel analysis.

The use of an automated verification system was an integral part of our verification approach. We chose to use the Gypsy Verification Environment (GVE), which is one of two verification systems endorsed by the National Computer Security Center. The GVE provides a collection of facilities for formal verification of programs expressed in the Gypsy language. These facilities include the following:

- a parser, which checks Gypsy text for syntactic and semantic errors;
- a verification condition (VC) generator, which generates theorems concerning a program's consistency with its formal specifications; and
- an interactive theorem prover which is used to prove these theorems (or VCs) by the application of mathematical logic.

In order to use the GVE to automate the task of proving FTLS consistency with the formal security policy model, we had to express both the model and the FTLS in the Gypsy language. We then had to construct a Gypsy program, in the form of a state machine, representing the KFM's operating environment. Within this Gypsy program, invocations of KFM kernel services were represented by procedure calls to the corresponding procedures defined in the FTLS. The effects of such invocations were represented by changes in a global state variable. These changes were translated, by means of a Gypsy function referred to as the interpretation function, into corresponding changes in an abstract protection state defined by the formal security policy model, thereby enabling verification of the state changes and the resulting protection states with respect to the properties of the model. The GVE's verification condition generator was applied to the Gypsy representation of the KFM's operating environment, in order to generate the necessary theorems for proving FTLS consistency with the model. These theorems were subsequently proved using the GVE's interactive theorem prover.

Other tools have been integrated into the GVE in order to reduce the amount of user interaction required to use the theorem prover and in order to provide automated support for covert channel analysis. These tools are the Gypsy Reprover, used to replay previously generated proofs, the Deductive Theory Manager (DTM), used to automate similar proofs, and the Gypsy Information Flow Tool (GIFT), used to automate the process of identifying covert channels. These tools are discussed in detail in the Lessons Learned section.

Lessons Learned

Many valuable lessons were learned in the course of this project. The three most valuable lessons for the research community concern prototyping, use of the DTM and Gypsy Reprover, and use of the GIFT. These are discussed in the following sections.

Prototyping Design Verification

The most important lesson we learned was that rework of both the system design and verification tasks can be reduced by prototyping the verification process. Problems in the verification approach can be costly to correct because they usually are not uncovered until late in the design process. The advantage of prototyping is that lessons learned from verification of one kernel service can be applied to verification of the other kernel services.

During prototyping, for example, we learned that the demonstrations of FTLS and DTLS consistency with the formal security policy model should be performed only *after* the covert channel analysis has been completed. At first, we did things the other way around; we performed the formal proofs demonstrating FTLS consistency with the model *before* we performed the covert channel analysis. As a result, we had to repeat a considerable number of those formal proofs when we made changes in the FTLS to eliminate covert channels. The reason we had to repeat so many proofs was that the kinds of changes that were necessary to eliminate covert channels (i.e., changes in data types) tended to impact multiple proofs in fairly drastic ways. By contrast, the kinds of interface changes that were necessary to correct inconsistences between the FTLS and the security model typically impacted only a few proofs and in relatively minor ways. We also discovered that more effort was required to repeat the proofs of FTLS consistency with the model than to repeat the covert channel analysis.

We concluded that we could reduce the overall cost of the verification effort by performing the covert channel analysis first. Consequently, for the remaining kernel services, we changed the order of the verification tasks.

Another lesson that we learned concerning prototyping of the verification process was that it provided an excellent vehicle for training. We found that in the course of the prototyping effort, our novice verifier learned nearly everything he needed to know about verification tools and techniques. He was able to complete the design and verification of the remaining kernel services with little or no help from the verification consultant. This result suggests that a larger proportion of inexperienced personnel could be employed in future verification efforts and that they could be trained by means of prototyping exercises. Such an approach would do much to promote the spread of verification technology.

Finally, we learned that prototyping can be used to promote the effectiveness of the DTM tool as described in the following section.

Use of the DTM and Reprover

The Deductive Theory Manager (DTM) is used to automate some common sequences of proof steps, in order to reduce the manual effort required to complete the formal proofs of FTLS consistency with the model. These common sequences of proof steps may be either complete proofs or portions of a proof. The information necessary to automate the proof steps is encoded in a knowledge base. It is encoded in the form of general rules and scripts which can be applied to a wide variety of similar proofs.

The DTM is most effective when many of the proofs share a common proof strategy. Once a common proof strategy is encoded in the DTM's knowledge base, the DTM can automatically perform proofs for which that strategy is appropriate. The trick is to determine which proof strategies are worth incorporating into the DTM's knowledge base. Since a significant investment may be required to incorporate a proof strategy into the knowledge base, the cost savings from automation of the applicable proofs must be sufficient to justify the investment in engineering the knowledge base. Prototyping provides an opportunity to formulate various proof strategies and to determine which ones are most suitable for incorporation in the DTM's knowledge base.

The most important lesson that we learned concerning use of the DTM is that the investment required to incorporate a proof strategy into the knowledge base must be carefully weighed against the potential for reuse of the proof strategy. One example of where we found this investment to be worthwhile, for the KFM, was in automating the top-level proof of each kernel service. It is interesting that we were able to make effective use of the DTM at all, since our example was intentionally chosen to demonstrate a diversity of proof strategies. The DTM is most effective when there are large numbers of similar proofs.

The Gypsy Reprover (sometimes referred to as just the Reprover) tool is used to automatically repeat previous formal proofs after correcting errors and omissions in the model and the FTLS, and after modifying the FTLS to eliminate covert channels. One part of this tool, the Command Extractor (CE), extracts theorem prover commands from a proof log produced by the interactive theorem prover in a previous proving session. The commands extracted by the CE are written into a command file. This command file is then supplied as input to a second part of the Reprover tool, the Command File Executive (CFE), which invokes the interactive theorem prover and then replays the commands.

The most important lesson that we learned concerning use of the Reprover is that it can be extremely useful in revalidating proofs after making changes to the FTLS. The Reprover is able to automatically revalidate any proofs that are not significantly affected by the changes. It is often possible to use the Reprover to repeat some portions of a proof and to complete the rest of the proof interactively. Sometimes it is also possible to anticipate the changes that will be required in a proof and to manually edit those changes into the command file before invoking the Reprover's Command File Executive. We found that, through a combination of these techniques, we were able to use the Reprover to reprove previously generated proofs approximately 75% of the time.

The Reprover is capable of repeating only the exact same proof steps that were performed in an earlier proof. If the changes in the FTLS affect the operations that are required for the proof, the Reprover cannot repeat the proof. Even if the required operations are identical, any changes in the number or order of hypotheses appearing in the theorems and/or the number or order of variables may adversely affect the Reprover's ability to repeat a proof.

Another lesson that we learned concerning the Reprover is that certain precautions must be taken during an interactive theorem prover session in order to ensure that the proof will be repeatable using the Reprover. For instance, justifications must be provided immediately for any *claim* operations (i.e., the *claim*ed fact must not be used until after it is proved). When a *claim* operation is performed, the person con-

ducting the formal proof has the option to justify the *claim* immediately or postpone the justification until the end of the current branch of the proof. The Reprover tool cannot repeat the proof if the latter option is chosen, since it cannot locate the proof steps for the corresponding justification in the proof log.

Use of the GIFT

The Gypsy Information Flow Tool (GIFT) automates the application of formal flow analysis techniques to the FTLS, in order to identify covert channels. Specifically, the GIFT is used to automatically generate a shared resource matrix (SRM) based on the formal specifications of the kernel services (from the FTLS). Once the SRM is generated, a security level is associated with each shared resource by means of a level function association set (LFAS). The LFAS associates a specified Gypsy function, whose domain is of the security level data type, with each resource. The GIFT is then used to generate security verification conditions (SVCs) for each potential information flow indicated by the SRM.

Initially, an SVC is generated for each potential information flow indicated by the SRM. Each SVC is a formula which may or may not be a theorem. If an SVC is a theorem, then its corresponding information flow is guaranteed to satisfy the information flow policy.

The most important lesson that we learned with respect to use of the GIFT is how to minimize the number of SVCs generated. The number of SVCs generated by the GIFT (i.e., the number remaining after automatic consolidation and simplification) is the single most important factor in determining the level of effort required for covert channel analysis. Each SVC must be analyzed by inspection, which is a very labor-intensive and potentially error-prone process. Thus, as the number of SVCs increases, the amount of labor required for their analysis also increases, as does the opportunity for making mistakes in their analysis. In addition, if any of the SVCs are to be formally proved (as was done for the KFM), the formal proofs will require a considerable amount of labor. The amount of labor required for the formal proofs depends on the number of SVCs to be proved, which also increases along with the number of SVCs generated.

All of the techniques for reducing the number of SVCs described below are required to make the number of SVCs small enough to make the GIFT usable.

We learned that one way to reduce the number of SVCs generated by the GIFT is to follow certain guidelines for the Level Function Association Set (LFAS) supplied as input to the GIFT:

- Assign the system-low security level to all kernel state components that are not modified by any of the kernel services and do not contain classified data;
- Assign the system-high security level to all kernel state components that are not read by any of the kernel services;
- Assign the subject security level to all kernel state components associated with a particular subject;
- Assign the object security level to all kernel state components associated with a particular object;
- Assign the system-high security level to all kernel state components associated with inactive subjects, inactive objects or invalid authorizations;
- Assign the subject security level for the currently executing subject to the User resource.

These assignments of security levels to system resources seem to significantly reduce the number of SVCs generated by the GIFT. They are also easily determined from the shared resource matrix (SRM) generated by the GIFT.

We found that we could further reduce the number of SVCs by configuring the GIFT settings as follows prior to dependency analysis:

- SET GIFT EXPANSION SVCS
- SET GIFT MODEL-LOW <system-low>
- SET GIFT MODEL-HIGH <system-high>

(where <system-low> and <system-high> are the minimum and maximum values, respectively, of the security level type defined in the FTLS). The GIFT EXPANSION setting controls the degree to which functions appearing in the SVCs are expanded by the GIFT. The default setting does not expand functions appearing in the SVCs. Consequently, many different SVCs are generated. The SET GIFT EXPANSION SVCS command causes all functions appearing in the SVCs to be fully expanded. When fully expanded in this manner, many of the SVCs turn out to be *isomorphic* (i.e., there are many duplicates). The GIFT automatically consolidates isomorphic SVCs (i.e., it eliminates duplicates), thereby reducing the number of SVCs by an order of magnitude. After consolidating isomorphic SVCs, the GIFT automatically simplifies each SVC using the simplification capabilities of the GVE's interactive theorem prover. The SET GIFT MODEL-LOW and SET GIFT MODEL-HIGH commands allow the GIFT to use lemmas about the

system-low and system-high security levels during the automatic simplification phase. In particular, the GIFT uses these lemmas to automatically eliminate any SVCs of the form x dominates system-low or of the form system-high dominates x. This further reduces the number of SVCs generated by the GIFT.

In analyzing the SVCs generated by the GIFT for the KFM, we found that the "guard" SVCs usually were not indicative of covert channels. These SVCs appear to be the result of overly conservative assumptions on the part of the GIFT with respect to flows of information from guard conditions. The GIFT assumes that information always flows from guard conditions to resources whose values depend on the guard conditions. For example, consider the *if* expression shown below:

```
if G then
  T = a
else
  T = b
fi;
```

The concern is that a user may be able to infer something about the guard expression, G, based on the value of the target resource, T. If any of the resources appearing in the guard expression are assigned a higher security level than T, there is a potential covert channel. The assumption that information always flows from G to T is overly conservative, however. It may be that the change in T is not observable by the user or that it is only observable in one of the two cases (i.e., when G is true or when G is false, but not both). Such is the situation for the KFM, when T is a component of the kernel state. Because the error conditions result in no observable change in the kernel state, one cannot infer anything about the resources used to test for errors by observing changes in the kernel state. An observable change in the kernel state only occurs in the absence of an error; but in that case, the access checks guarantee that observation of the change in the kernel state does not violate the security policy. When T is not a component of the kernel state (e.g., when it is an output parameter for the kernel service), though, the "guard" SVC may indeed indicate a covert channel.

In writing the FTLS, we found that any functions used (either directly or indirectly) in the specifications of the kernel services should conform to the following style:

- With few exceptions, they should not be declared as *pending* (the exceptions are functions that do not have any part of the kernel state as a parameter and that do not produce components of the new kernel state);
- Whenever possible, they should be written in a form that is expandable by the interactive theorem prover (i.e., they should be of the form *result* =< *whatever* >);
- The parameters of any non-expandable functions should not include any unnecessary components of the kernel state.

This style was suggested by our verification consultant and is also recommended in Appendix D of the *ASOS Covert Channel Report*[TRW90]. It is intended to facilitate formal flow analysis using the GIFT. If the FTLS does not conform to this specification style, the GIFT will either abort during dependency analysis or it will generate a very large number of SVCs. Note that this specification style need not be imposed on any of the interpretation functions used to map components of the kernel state into components of the model's protection state. Also, note that this specification style need not be imposed on any of the other functions used in the Gypsy representation of the KFM state machine.

When performing dependency-analysis, we found that the GIFT may complain about functions that it cannot expand. Either these functions must be reformulated in such a way that they are expandable or their parameter lists must be changed so that only a minimal portion of the kernel state is passed to each function.

The results of our covert channel analysis are summarized in the following table. Each kernel service is listed with the number of total SVCs generated, the number of SVCs which we were able to prove, the number of SVCs we were unable to prove and the remaining covert channels. We added a delay to the Write Disk kernel service to reduce the bandwidth of the remaining two covert channels.

Summary of Covert Channel Analysis Results

Kernel Services	Total SVCs	Proven SVCs	Unprv SVCs	Covert Chnls
Create File	22	18	4	0
Delete File	27	12	15	0
Open File	6	5	1	0
Close File	2	2	0	0
Read Disk	3	3	0	0
Write Disk	14	10	4	2
Modify DAC	49	22	27	0

Additional Lessons Learned

A few of our other valuable research lessons learned are described in the following paragraphs. Each paragraph describes a single lesson learned.

We found that the formal verification process identified flaws in the requirements and design. From

mapping our requirements back to the TCSEC we discovered we were missing requirements to address object reuse. We needed to impose some additional requirements to meet the object reuse criteria and to provide greater control over propagation of discretionary permissions. For example, we added a requirement for the Close File kernel service to reclaim the specified file object so that it may be reused. We also added a requirement for the Modify Discretionary Access kernel service to ensure that modification of discretionary access permissions would be allowed only if the calling program instance were operating on behalf of the owner of the specified file.

In developing the interpretation function, we found it useful to decompose the function into a hierarchy of functions based on the structure of the protection state data type. This is the way we initially decomposed the interpretation function for the KFM. Since the protection state consists of four components, we decomposed the interpretation function into four functions, each of which returned one of the four components. Each of these functions took the corresponding portion of the kernel state as a parameter.

We found it essential to save core images that could later be re-executed, rather than using the GVE's *save* and *restore* commands. The GVE's *save* and *restore* commands save the GVE's internal database in a file and restore it from a file, respectively. Each time the GVE's database is saved and restored, the database grows in size (this is a known bug in GVE version 20.70). This can lead to a fatal error during restoration of the database, since the GVE eventually reaches its dynamic memory allocation limit. An alternative way to save the GVE's internal database is to save a core image (i.e., an entire memory image) of the GVE in execution. This is done by invoking the LISP environment upon exiting the GVE, and entering the LISP command "(si::save-system "<filename>")" (where <filename> is the name of the file in which the core image is to be saved). Another benefit of saving core images is that the time required to save them and reexecute them is considerably less than the time required to execute the GVE's *save* and *restore* commands.

We learned not to accumulate completed proofs in the GVE's internal database. As proofs accumulate, the size of the GVE's internal database grows. As the database grows, the performance of the GVE's interactive theorem prover degrades severely. This problem can be solved by using a freshly parsed database for each proof (we saved a core image, to avoid having to reparse the specifications each time).

We discovered that it is effective to divide the FTLS into multiple source files in such a way that each individual kernel service can be parsed and proved separately. Parsing and proving only a portion of the FTLS at a time minimizes the size of the GVE's internal database, thereby optimizing performance. Dividing the FTLS into multiple source files also facilitates maintenance and configuration management of the FTLS.

One of our most important lessons learned was actually relearned. We found that lemmas were valuable in modularizing the proofs and improving performance of the GVE.

Benefits and Cost

Our experience demonstrates that there are many benefits to using formal verification technology. First, the verification process uncovers problems in the requirements and design. If a prototyping approach is used, these problems can be surfaced early in the project life cycle to save cost and schedule. Secondly, there are benefits from writing the formal top level specifications. This exercise helps define the kernel services precisely, especially with respect to error conditions. Writing the formal specifications also encourages the definition of simple interfaces and provides the basis for test case generation. Finally, formal design verification has many benefits. Covert channels are detected early in the project life cycle, assurance is gained that the design implements the security model and requirements and design flaws are uncovered.

There is of course some cost to perform formal verification. We estimated that, for the Kernel File Manager, the formal verification process added about 40% to our development estimate. Although we did not have requirements to code and test the KFM, we estimated the cost to do so. We assumed that it would take 3000 lines of code to code the KFM. At a productivity rate of 75 lines per man month, it would take 40 man months to code the KFM. It took 16.3 man months to perform the formal verification tasks for the KFM (we subtracted out 1.5 man months of design time from the 17.8 man month total). Dividing the time to perform verification by the time required to develop the KFM (16.3/40) gives 40% as the overhead to perform verification of the KFM.

Of that 40% it is interesting to note where we spent our time. 15% was spent writing the model and another 8% was spent proving the model was internally consistent. 14% was spent writing the FTLS and 47% proving the FTLS was consistent with the model. 16% was spent performing the covert channel analysis. What is interesting about these numbers is

that we got the least benefit from proving the FTLS was consistent with the model, which represents the largest percent of our time. The only requirements and design flaws were uncovered during covert channel analysis. It might be more cost effective to, in addition to proving the model is internally consistent, prove only selected (by the customer) portions of the FTLS and to write up English descriptions of the proofs for the remainder of the system.

We think the 40% overhead number is a worst case number. The 23% of the time spent on the model was essentially a one time cost. If this example were to be extended, the time to update the model would be minimal. If the 47% spent on proving the FTLS is consistent with the model could be reduced by proving only selected portions of the system, we think the 40% overhead could be reduced to 20% or even 15%.

Conclusions

The Current ETL Examples project has produced a a simple, yet realistic, worked example of A1 verification technology and produced many valuable research lessons learned in the process. The most important of these to the research community are the following:

- Prototyping the design verification process reduces rework of both the system design and of the verification tasks.
- Using the Deductive Theory Manager and Reprover tools make the Gypsy Verification Environment user more productive.
- By applying a combination of techniques, the Gypsy Information Flow Tool can be used on a significant example.

Acknowledgements

The authors gratefully acknowledge the review comments of Karen Ambrosi, our contract monitor, Michele Pittelli who attended all of our project reviews and made many helpful comments, Jim Williams, our MITRE reviewer, and of our colleague Ruth Hart. We are also indebted to our formal verification consultant, Alex Murray. This paper is largely the result of work conducted as part of the Current Endorsed Tools List (ETL) Examples project, sponsored by the Department of Defense, under Contract No. MDA904-90-C-7058.

References

[BLD90] B. L. DiVito, P. H. Palmquist, E. R. Anderson and M. L. Johnston, "Specification and Verification of the ASOS Kernel", *Proc. 1990 Symposium on Research in Security and Privacy*, IEEE, May 1990.

[BDCG90] B. DiVito, C. Garvey, D. Kwong, A. Murray, J. Solomon, and A. Wu, "The Deductive Theory Manager: A Knowledge Based System for Formal Verification", *Proc. 1990 Symposium on Research in Security and Privacy*, IEEE, May 1990.

[Cen85] National Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, DoD-5200.28-STD, December 1985.

[JM89] J. McHugh, R. L. Akers and M. C. Taylor II, *GVE Users Manual: The Gypsy Information Flow Tool, A Covert Channel Analysis Tool*, Technical Report #12-c, Computational Logic, Inc., Austin, TX, July 1989.

[DoD88] Department of Defense, *Security Requirements for Automated Information Systems (AISs)*, DoD Directive 5200.28, March 1988.

[TRW90] TRW, *Covert Channel Analysis Report for the Army Secure Operating System (ASOS)* (Final), CDRL G007, delivered under contract DAAB07-86-CA032, TRW Systems Integration Group, September 1990.

[TRW90b] TRW, *Software Users Manual for the Gypsy Reprover Software Development (Reprover)* (Final), CDRL A001, delivered under contract 731933-FS, TRW Systems Integration Group, March 1990.

[TRW91] TRW, *Formal Security Policy Model for the Current Endorsed Tools List (ETL) Examples* (Final), CDRL A003, delivered under contract MDA904-90-C-7058, TRW Systems Integration Group, August 1991.

[TRW91a] TRW, *Final Report for the Current Endorsed Tools List (ETL) Examples* (Final), CDRL A004, delivered under contract MDA904-90-C-7058, TRW Systems Integration Group, August 1991.

DATA SECURITY FOR PERSONAL COMPUTERS

Paul Bicknell
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730
617/271-3625

ABSTRACT

Personal computers are vulnerable to a wide range of risks including viruses, trojan horses, and damage by malicious or inadvertent use. These vulnerabilities are based on the lack of integral security provisions designed into PC operating systems. A number of models of security have become available to address this lack in PCs and this paper presents these by establishing identified model groups and by discussing the security technology represented by each group. Each model group provides a unique solution to a particular PC security vulnerability.

INTRODUCTION

The use of personal computers (PCs) has become a significant factor in data processing. PCs are rapidly replacing the reliance on large centrally located time-sharing systems for general purpose computing in offices and laboratories. As this transition to locally based computing continues, attention is being focused on the issue of providing adequate protection for data stored on these PCs. These systems have a near total lack of inherent data security provisions and are vulnerable to data loss and corruption from either inadvertent or malicious actions. The basis for these threats lies with the absence of integral mechanisms, which could limit the actions of a user on stored data resources. These systems have very little ability to deny a user, or program running on behalf of a user, any operations that the system is designed to perform.

As PCs have become widely used, a trend to view these systems as additions to, or actual replacements for, time-shared, multiuser, mini- and mainframe computer systems has started. But as users transition away from the dependence on traditional systems, there has been a lessening of reliance on the security mechanisms of these larger systems to protect data assets. Mechanisms such as access control, auditing, and authenticated login are generally lacking in PCs, and this is introducing an increased risk to the work done on these systems.

The vulnerabilities of data stored on PCs has been apparent for some time to a group of programmers who are independently developing hardware and software packages intended to be added to existing PCs to partially resolve the security deficiencies. These developers are working with systems that are initially wide open and are designing innovative solutions to the security problem. However, due to the lack of a native security models in PCs, there is no standard security architecture to limit the solutions these developers are producing. For this reason, a large number of add-on security packages have become available which provide a

variety of dissimilar security technologies to address the most common problems of physical protection, access control, and defense against computer viruses.

The intent of this paper is to present the results of a study into available security technologies and to describe the logical groupings into which the majority of existing add-on security systems can be organized. It does not seek to give examples of how to build a security system; rather, an attempt is made to provide insight into how others are currently designing security packages and to describe the security model of each architecture.

The Lack of Security in PCs

The primary reasons for the lack of inherent security provisions in PCs are economic and technical. The economic reasons are quite evident since the goal to make an inexpensive system would have been compromised if the operating system and hardware were designed to provide data protection. The technical reasons are closely tied to the economic and relate to the availability of affordable mass-storage devices when PCs were being conceived in the 1970s.

The lack of hard disk drives necessitated the use of removable floppy disks which meant that permanent data storage on a PC was not possible. There was little design justification for introducing security mechanisms into an otherwise simple operating system.

There were other reasons why the original PC designers avoided security, and these were related to a general revolution in computer usage that occurred as a result of the advent of the personal computer. This revolution had a definite goal, which was to bring *computers to the masses*; to do this required an easy-to-use, nonintimidating, user interface. PCs were going to be marketed as appliances; and under this view, to protect one user's data from another's was pointless because each user would have his own isolated system. In this situation there was no need, for what to a nonuser would seem a threatening confrontation with a computer, to require the user to identify and authenticate himself before beginning to perform any operations.

Problems existed in reasoning against security and included not anticipating the rapid development of cost effective hard disk drives and the long-term storage of data on systems. Also not considered was another equally rapid advance in technology that occurred at the same time as the PC revolution, networking. Advances in communications technology allowed the previously isolated PCs to be connected together through networks and telephone lines.

The irony is that what initially consisted of multiple users sharing a common resource (a time-sharing system) has not changed. Multiple users are still sharing a common resource (the communications network and all the PCs connected). Only now, unlike a time-sharing host which had adequate security (e.g., authenticated login, access privileges, etc.), the current networked distributed systems made up of PCs have very little if any data protection capabilities.

The basis of the security vulnerability of PCs was fundamentally related to the architecture of the original processor chips. The processors were single state machines. That is, there were no supervisor state or privileged commands to provide a security barrier around powerful operations. There was no limit to or restrictions on the actions of users on stored data resources, and users could not be denied total access to the system. Furthermore, users were

able to run *private* versions of the operating system by booting from the floppy disk drive, so any modifications made to the resident operating system could be overcome. Also, low level utility tools became widely available, allowing users to perform *raw* I/O instructions beyond the control of the operating system.

To alleviate these security deficiencies, software/hardware packages have been developed. However the lack of any standard security architecture has resulted in the creation of numerous designs. These fall into three fundamental security model groups: physical protection, domain isolation, and viruses and Trojan horses.

PHYSICAL PROTECTION

Physical security is conceptually the simplest way to protect stored data. The primary objective is to secure a PC during the time a user is not active on the system and thereby prevent access by other users either to the machine or to the data stored on the machine. No attempt is made to share resources, and no protection is provided against other machines accessing across a network. Also, no attempt is made to provide any sort of interactive data protection - the PC is either totally open or completely closed. In addition, few or no mechanisms are provided to protect a PC against the introduction of a computer virus which may be carried into the system hidden in a useful program.

Two subgroups of physical protection systems exist: access lockout and removable components.

Access lockout protection systems are those which attempt by physical means to totally lock down a system when a user leaves, These systems are usually key-locking systems which can provide such protection as making the power switch inaccessible or preventing keyboard commands from reaching the processor. They can also physically attach the protected PC to a table, thereby preventing its removal. These protection systems also include lockable cabinets in which the entire PC can be stored.

Removable component protection systems provide data security by storing the data on devices which can be physically removed from the PC and kept away from any user who might want to steal or damage the data. The original PCs incorporated this protection scheme by having all data stored on floppy disks which could be locked up when a user wanted to protect data. More recent systems include entire disk drives which can be removed and large data cassettes (e.g., the *Bernoulli Box* system) which hold more data than floppy disks. These systems typically do not prevent access to a PC and can, in fact, allow a machine to be shared by multiple users. They do, however, allow users to segment their data and prevent others from accessing it, but they provide little protection from viruses.

DOMAIN ISOLATION

Domain isolation security systems are more complex than the physical systems and serve to protect data files which are permanently stored on a PC from other users or programs being executed. These protection systems generally attempt to protect *passive* resources, such as data

files stored on disk or in memory, from modification or destruction by *active* agents, such as executing processes, without otherwise interfering with the execution of the process. They provide mechanisms to allow a separation of data into private domains for storage, permitting multiple users to share a single machine by isolating individual user's resources.

To provide this protection requires the concept of permitted vs. nonpermitted operations, and this necessitates the abstractions of identifiable users and privileges. This is typically accomplished by introducing the model of multiple users and ownership of resources into what was originally a single-user system. Once this is done, it is possible to define permitted and non-permitted actions by associating certain privileges, either implicit or explicit, with users.

Implicit Privilege

Of the types of domain isolations systems, implicit privilege are the simplest, conceptually. They are designed to allow or disallow access to protected resources not by differentiating the identity of an individual user from all other users but by associating access to a resource with possession of some kind of token whose existence is coupled to the protected resource. For these systems, the possession of the token *implies* that the user has the necessary privileges to access the resource.

The chief examples of this type of protection system are those that use encryption to deny nonprivileged users access to a protected data file. Here, the knowledge of the encryption password amounts to possession of the token which implies that the user has the necessary access privileges. Other examples of this type of protection system are *smart cards*, cryptoignition keys, and various additional systems using things such as specially formatted boot disks. For any of these, the protected PC may be otherwise accessible to any user, but a subset of the data resources are stored in encrypted form and are accessible only to the set of users with the knowledge of the password or the possession of the physical token which may contain the encryption key.

Two types of encryption systems exist: direct and transparent. Direct encryption systems are the simplest and usually store protected files by encrypting them with separate passwords. Each time a user wants to access a protected file he supplies the corresponding password with a command and the protection package performs the decryption (or encryption). These systems require very few modifications to a PC's operating system, and the security package exists as an application program.

Transparent encryption systems are more complicated and require either an additional hardware board or a more complicated program. These systems require only an initial password either typed in by the user or contained on a smart card or cryptoignition key. They then automatically encrypt/decrypt all data references to the hard disk. The users are typically unaware of the data encryption and only see plain text displayed on the screen. If, however, they used a low-level I/O program to do direct disk access, they would find that all of the data in a protected file was encrypted. These encryption systems require that all I/O operations to the hard disk be interrupted, but this can be done without having to modify the existing operating system.

Encryption systems are the simplest form of this category of protection, but there are others which use passwords to limit access but that do not store the protected files in encrypted form. These systems have software mechanisms which interrupt operating system access operations and require the user to enter a password. This password is then compared against a stored list of passwords and protected resource names, and, if a match is found, the operating system is allowed to continue the operation. These security systems require the creation of password storage data files, the user interface necessary to create and maintain the data files, and the protection of those data files.

Explicit Privilege

Explicit privilege security systems are the most complex examples of the domain model. They require a much more complicated program with low-level interfaces into the PC's operating system and usually result in the implementation of a fully functional multiuser environment, although usually only one user can be active at any time. Like the implicit privilege group, they protect static data files from active users or processes. However, unlike most of the implicit privilege systems, these types allow or disallow actions based on a specific user identity. To do this requires the implementation of the double abstraction of privileges and permitted/denied operations and the adoption of the discretionary access control security model.

To implement a multiple-user system requires the maintenance of complex data structures and special files containing the information necessary to establish user identities (i.e., user names and passwords). To assure security, this database must be protected from user access, and a protocol of permitted access to this database must be established to allow users to be added or removed. Further, to make use of this database, a *login shell* or other mechanism must be added to verify and authenticate the identity of any user attempting to use the system. These mechanisms must also be able to lock a user out if he fails to supply the correct identity information.

In addition to identifying users, an explicit privilege system must be capable of permitting or denying actions invoked by users. This necessitates the introduction of specific access capabilities and the linking of those capabilities to identified users and protected resources. This typically requires the addition of an *access control* database which contains a mapping of user identities and protected resources coupled with the definition of specific access privileges. To utilize the access privileges, a new operating system capability must be introduced which allows access attempts to be mediated and the database to be checked before allowing or disallowing the attempt.

In addition, a specially privileged user must be defined. This user, the system administrator, or security officer must be provided with a separate command interface to the security system and must be responsible for configuring the access control database and monitoring its integrity. This user's command interface must allow access to operations which can add or remove users and can grant or withdraw access privileges to specific resources. This user is also responsible for routinely checking the security state of the system. This is usually accomplished via audit data, another mechanism added with this type of security system.

There are two types of explicit privilege security systems, and they differ based upon the relationship of access verification and access attempt. The two types are proactive and reactive.

Proactive Explicit Privilege

Proactive explicit privilege systems make access validity determinations before any operations are attempted. They place the system in a prearranged state where the user is preverified for all available operations. These security systems tend to use whole screen windowing where actions are invoked by cursor positioning.

In these types of security systems, the access control database is checked and all access mediation decisions are made when a user logs in or changes state. Access is controlled *proactively* by the format of the interface presented to the user. The format of any screen is a function of the user's access rights. The decisions about what operations a user can perform on which data objects have already been made by the time the user has any control of the system. Permitted actions and permitted data resources are usually pictorially displayed in a formatted window.

The system administrator is responsible for configuring each screen format for each user and then entering this definition into the access control database. This database will be referenced each time a user logs in or changes state (e.g., exits an application program), and the appropriate window will be constructed and displayed. The assurance of these systems rests on prohibiting ordinary users from exiting the access controlled windows and establishing contact directly with the PC operating system.

Reactive Explicit Privilege

The other type of explicit privilege security system, reactive explicit privilege, is the most complex of all the PC security systems. These systems make access control decisions before every operation invoked by a user or program running on behalf of a user is allowed to proceed. They provide the most conventional security model where access control decisions are based on user privileges and where the operations are being attempted on specific data objects. A fully functional multiuser access controlled environment can be constructed, although most of these systems can support only one active user at a time.

To implement one of these systems requires not only the same access control databases as used by the proactive group but also much more complicated interfaces and modifications to the underlying operating system. For these systems it is necessary to add the code to verify each access request immediately before it is handled by the operating system and also to add error and denial logic code paths to process rejected access attempts.

These security systems operate *reactively* and typically interface at the device driver level to the PC operating system. In this way, they can capture data object reference operations just before they are dispatched to the device driver and can perform an access control determination based on the identity of the user, the name of the file object, and the type of operation being attempted. If access is permitted and the operation is allowed, control is passed to the device driver and the operation proceeds accordingly. If the operation is not permitted, error messages can be displayed and audit records written. Control can be given back to the operating system so it can exit via an error return path.

These are the most complex security access control systems, and they can be made to operate in an unobtrusive manner with a small access monitoring kernel operating in the background. Users interact with the operating system in a seemingly normal fashion once they have logged on and supplied a validated user name and password. Only when an access attempt fails for a security-relevant reason will the user be made aware of the operations of the security system. Otherwise, the behavior of the protected PC will appear normal, although performance may be affected to some degree.

Like the implicit privilege security systems, the explicit privilege systems require a means of self-protection. However, the protection for these systems is more complicated since users are granted full access to operating system commands and can invoke editors and debuggers, etc. In order to protect itself and the access control data files, the explicit privilege system needs complex mechanisms. This includes such things as data checksum validation to detect manipulation attempts of both access control data files and the security critical regions of the protection system itself.

The protection mechanisms may do other things such as monitoring the system clock to detect attempts to deactivate the security system. This way, a determination can be made, albeit after the fact, of a possible security violation and appropriate actions such as sounding alarms or halting the PC can take place. In either case, the security administrator will be alerted that a violation has occurred and can handle the situation accordingly.

A security system also needs to defend against the threat posed by debuggers by understanding how they function and detecting their execution before any security violation can occur. However, one threat that any of the domain isolation systems have less success in defending against is that presented by computer viruses.

VIRUSES AND TROJAN HORSES

Viruses and Trojan horse programs present a difficult threat for PC security because they necessitate a security model very different from the conventional and widely implemented Discretionary Access Control (DAC) model. The reason for this is that the objective of DAC is to protect data resources owned by one user from access attempts by a separately identified user. This security model does not hold for viruses since the threat is not posed by another user directly but indirectly via a program which contains a hidden *bomb* that can be triggered when any user runs this program. When this rouge program is executed by an unsuspecting user, it will have all the necessary privileges to damage or destroy all the data resources accessible by the user.

To counteract this threat requires a security model different from the domain isolation models where passive resources (files) are protected from active agents (executing programs). This model must provide for the protection of active resources from active agents by the security system specifically interfering with the execution of the agent itself. However, this is difficult to do because of the problems inherent in differentiating normal program behavior from aberrant program behavior. Fortunately, there are certain characteristics displayed by the preponderance of viruses which can be used to defend against them.

Virus programs have been detected in a large number of forms and have been observed to cause damage in numerous ways. But despite the number of types of viruses, they all share certain properties and behaviors which can be defended against. First, viruses and Trojan horse programs are usually transmitted to an unprotected machine as part of an otherwise useful program. They are carried along and installed when an unsuspecting user loads a new program on his machine. Second, the virus which has now infected the machine lies dormant until some triggering event takes place. This event may simply be the executing of the infected program at which point the virus can seize control and have unrestricted access, or it may be related to a preprogrammed time or date. Third, once triggered and made active, these programs can enter a propagation stage where they spread the infection to other programs on the PC. Fourth, while in the active stage, viruses can cause damage by doing things such as deleting files, modifying interrupt vector tables, disabling memory, decreasing system performance by introducing delays after every operation, etc.

To protect a system against viral infection, it is necessary to interfere with the virus at one of these stages by either preventing the virus from gaining entry to the system or by preventing the triggering or propagation phases. Two types of antiviral programs have been developed for this purpose: proactive antivirals and reactive antivirals.

Proactive Antiviral Programs

Proactive antiviral programs are self-contained user application programs which are executed prior to the running of a viral infected program. Suspected programs are examined for any trace of a virus, and, if any are found, the program can be prevented from being loaded. This can interfere with the infection phase by detecting the presence of a virus prior to storing the program and preventing the virus from being introduced into the PC. It can also interfere with the propagation phase by preventing the infected program from being executed and thereby deny the virus control of the system. Viruses are inherently benign until execution.

These types of antivirals typically search for some recognized pattern that indicates the presence of a virus. They look for bit patterns, or *tags*, indicative of a particular virus, and they alert the user if any are found. Proactive antivirals are easy to use and can be periodically run to reverify the absence of any viruses on the system. These are also by far the most common type of antiviral programs currently available and are often referred to as virus scanners..

The drawback to these antivirals is that they can only search for tags known to the author of the particular antiviral program. They are not particularly successful in detecting new viruses.

Reactive Antiviral Programs

The other major category of antiviral protection systems are the reactive antivirals. Unlike the proactive group, these programs do not search for recognized tags, or bit patterns, but attempt to detect changes in the state of the system. This is done by either detecting some change in the system state subsequent to the execution of a viral infected program or by performing real-time activity monitoring. There are two distinct types of reactive antivirals: signature checkers and behavior monitors.

Signature checking antivirals search for a telltale change in the system state subsequent to process execution. They provide program and data file integrity checks that can determine if a change has occurred as a side effect of a process execution. This usually requires the establishment of a data record against which the system can be compared. This data record is usually some sort of checksum *signature* of a program's image which can be checked before or after program execution to detect modifications which might be caused by a virus.

If after execution a program's signature is found to be different, the program can be flagged and brought to the attention of the user. This action will interrupt the propagation phase of a virus since the newly infected program can be prevented from executing and further spreading the virus. The source of the virus will still need to be isolated.

The advantage of these types of antivirals is that they can be effective against all viruses which attack program executable images. They are not limited to being effective against only those viruses known by the author as are proactive antivirals. The disadvantage is that they are only effective against programs whose executable image is added to the database prior to the introduction of a virus. If the checksum database contains the image of an infected file, these antivirals will be unable to detect the virus although they can detect the propagation of the virus.

The other group of reactive antivirals are the behavior monitors which do *real-time* monitoring of program activities. These are the most unusual antivirals, and they attempt to differentiate normal program activities from those activities normally attributed to a virus. These antivirals are usually permanently executing processes which run in the background (e.g., DOS Terminate Stay Resident (TSR) programs) and observe other program's execution. This can interfere with viral propagation and damage phases by detecting programs attempting to seize certain system resources or performing write operations to system areas on the hard disk.

The advantage to behavior monitor antivirals is that they can be effective against large numbers of viruses, not just those known to the author. They can also catch viruses early and prevent the initial system infection. Their disadvantage is that they can often interfere with normal program execution which may routinely perform I/O, or other system activity, which the monitoring programs associates with viral activity.

CONCLUSIONS

Personal computers are known to have security vulnerabilities based on their architecture as single-state machines. A number of different security models have been presented which are capable of addressing these vulnerabilities and providing certain types of security ranging from physically locking a PC to prevent theft, to providing a multiuser discretionary access controlled environment, to monitoring program execution searching for computer viruses. Each of these types of security systems are successful in addressing the particular security goal addressed by them. They are less successful in addressing security issues outside their narrow focus. Antiviral systems, in general, make no attempt to identify individual users or provide controlled sharing of system resources. Physical protection systems provide little more than a way to lock up a PC at night, and the domain isolation systems provide little protection against viruses.

What is clear is that no single security model is adequate for completely solving the security problem in personal computers. To provide sufficient, meaningful data security for a PC, it is necessary to understand the threats posed by the intended use and environment and then to use some combination of examples of the types of security packages discussed here.

REFERENCES

CERTUS - User Manual; Foundation Ware Incorporated, Cleveland, Ohio.

Computer Security Subsystem Interpretation of the Trusted Computer System Evaluation Criteria, NCSC-TG-009 Version 1, 16 September 1988.

CRYPTOLOCK - User Manual; Commcrypt Incorporated, Rockville, Maryland.

FASTLock - User Manual; Rupp Corporation, Los Angeles, California.

Fred's Papers, Book 1; Fred Cohen, 1988.

INTRA-LOCK - User Manual; MIU Automation Incorporated, Markham, Ontario, Canada.

OnGuard - User Manual; United States Software Incorporated, Vienna, Virginia.

Security In Depth; *Byte* , June 1989; McGraw-Hill Inc., New York, NY

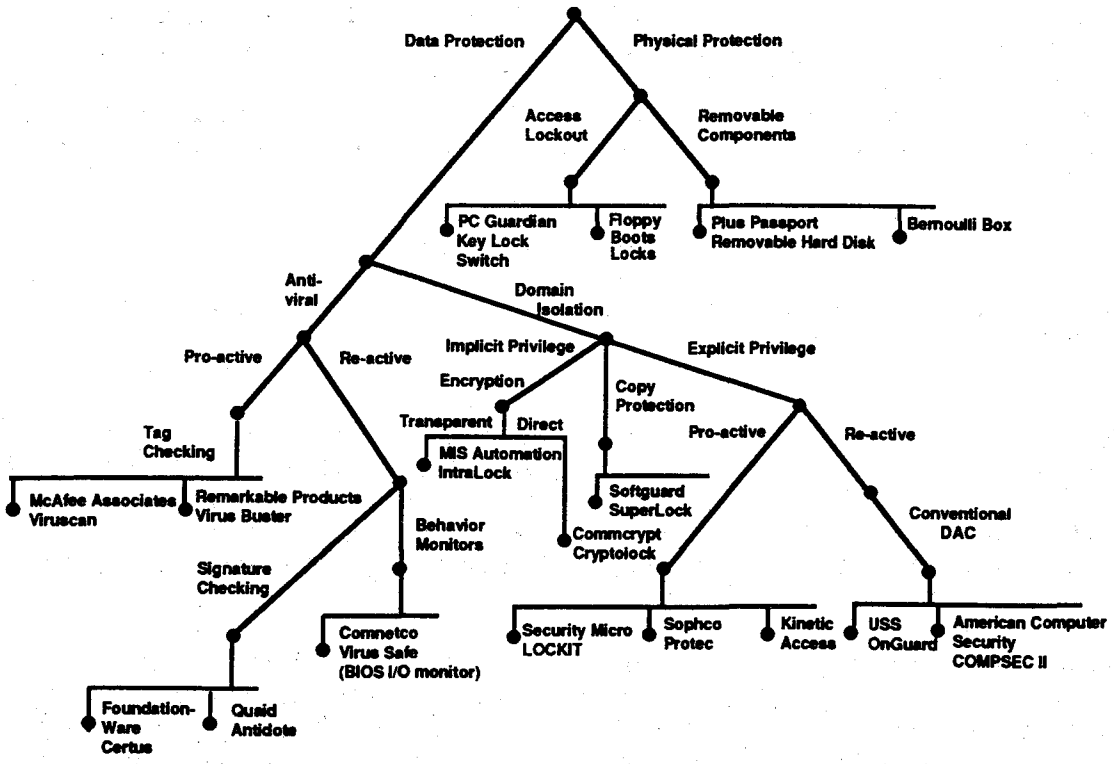


Figure 1 IBM PC System Security

DEFENSE AGAINST COMPUTER AIDS

MR HORACE B. PEELE
Chief, Security Division
Communications-Computer Systems
Air Force Intelligence Command
San Antonio, Texas 78243-5000
Phone: 512-977-2767

ABSTRACT

Computer Aids? Everyone knows that the Acquired Immune Deficiency Syndrome, commonly called **AIDS**, is a breakdown of the human body's immune system. Once contracted, a person with **AIDS** is subject to infection of almost any disease or virus and may finally die of such simple illnesses as the common cold. Everyone also knows that **AIDS** is reaching epidemic status.

Computer AIDS? Right! Why Not? Computers are manufactured without an immune system, therefore they do not need a breakdown to be subjected to infection by computer security illnesses to include the well known problems with computer viruses. So why not catalogue these computer security illnesses as **Computer AIDS**? This paper addresses the six common security threats which computer users must combat in order to have a good computer security program. Most users are not aware that the computer they use is a threat to their privacy, job, and their status.

BACKGROUND

These six security diseases have already caused many users to suffer the attacks of **Computer AIDS**, some with dire results. Each personal computer user must understand the results of these security diseases if he or she wants to ensure that their jobs or privacy have not been violated by such computer security diseases as the **"PRIVACY INVADER"**. Research on these six security diseases has been independently accomplished by many organizations and individuals. And some measures have been independently developed to minimize the potentially embarrassing results of their contamination. However, only the **United States Air Force Intelligence Command** has developed a vaccination against all six diseases. This vaccination is known as **"The Computer Security Toolbox"**.

A PERSPECTIVE

How did we get started on this venture into computer security? Let's look back about three years at the results of two true stories.

Case Number One. A fifteen year career officer within one of the military services made a conscious decision to take an illegal copy of a government purchased word processing program from his place of employment to his private residence, a simple copyright violation, also simple theft. Later, he gave this same copy to a relative who in turn gave it to a neighbor further compounding the copyright violation. Although it is wrong, the number of violations is not important to this story. Here is what happened!

One illegal recipient (copyright violator), another military member, was reviewing the word processing program with a binary editing capability probably in an attempt to change the information about the licensee to himself and to eliminate reference to the "losing" military organization. While doing so, this individual found classified information in a file after the end-of-file marker. After considering the consequences, the individual reported the security incident to the appropriate military authorities.

A formal investigation led to the original violator. The copyright violation was a bad enough blemish on an otherwise perfect military career, but because of the security violation the officer was offered a choice--either resign from the service or be subjected to a court-martial. The officer resigned. Sad as this story is, many of us are guilty just like the officer. I wonder what information we have taken home on diskette without knowing the potential damage we have done to our national security. Are you guilty? Has this **PRIVACY INVADER** invaded your privacy?

Case Number Two. This is another similar story which involved an enlisted service member. It seems that in the early days of government purchases of personal computers, the government did not buy sufficient numbers of commercial off-the-shelf software packages to match the need of the users. As a result, many people brought their own software into government office space, just to be able to get the job done.

This dedication of our military to get the job done is admired, however; such action is within itself another security violation. There is a policy against using personally owned systems, both hardware or software, to do government work. It is illegal. And there must have been many misuses of privately owned software, because this particular military department offered every violator amnesty and instructed all such patriots to remove their private software from government systems and to take it home! Now, here you are, a military member being told to take it home! You obey orders! You take it home! You copy your software onto a government owned diskette and you delete the original from the hard disk. You walk out of the office with carte blanche exit privileges!!!

Sometime later. You hold a super high security clearance within your organization and as part of the personal security program, you must take a recurring lie-detector test in order to maintain your clearance status. You are not wanting to fail such a test, it could mean the end of your job. So when asked "Have you ever taken government property home for personal use?"; you answer "Yes." You can't win for losing. They got you! And during the investigation, you are asked to surrender all repeat all of your floppy diskettes for examination.

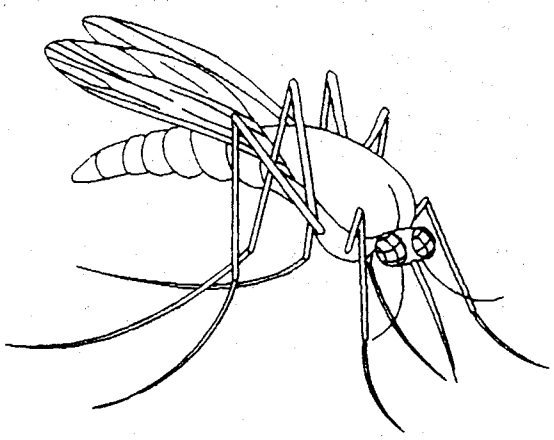
The examination revealed that you were clean of all problems except one! Remember that brand new diskette used to take your personally owned software back home? Well, it contained portions of a highly classified time sensitive intelligence report--national security information. The **PRIVACY INVADER** strikes again! But, because the diskette had never left your possession and since you had cooperated with the authorities, the investigator ruled in your favor. The diskette probably had not been exploited so there probably was not an actual security compromise, just a serious security deviation. Consider being a civilian working for a corporation with tons of proprietary information on high tech developments, the result could be devastating! But how did these two incidents happen?

DISCUSSION

What are these SIX computer security diseases--Computer **AIDS**? At this point the author invokes the right of authorship. For the purpose of this document, these diseases are the **PRIVACY INVADER, MISS CLASSED, ILLEGAL OCCUPANT, DAA BLESSING, CHEAP USER,** and **BIT DEATH**.

PRIVACY INVADER

As casual computer users, we know the basic reasons for the **CONFIG.SYS** and **AUTOEXEC.BAT** files. We know that they provide a communication path between the application software and the Disk Operating System (**DOS**) and that without them the system will not function. We also know that the statement **"BUFFERS = 20"** found in the **CONFIG.SYS** file



causes **DOS** to allocate 20 buffers of memory each time the system is powered up to be used for file management. These buffers are usually 512 bytes of memory and are used to control access to all files used by the applications which you activate. However, as casual users, we do not understand exactly how **DOS** uses these buffers. But for the purpose of this paper the following conception scenario is considered correct.

Misuse of these buffers by **DOS** activates the **PRIVACY INVADER** disease. Lets explain how this happens. First of all, when a system is powered up, the **COMMAND.COM** file is automatically loaded into memory to control the system. The buffers allocated by the **CONFIG.SYS** file are used to retrieve the **COMMAND.COM** file from the disk and to load it into operating memory. By the time you begin using the computer, many of the buffers will already contain residue program executable code from the **COMMAND.COM** file. Why? Simply because **DOS** does not clear buffers after their use.

So, lets say that the first application program you activate is a word processor. It gets loaded into memory through these same buffers. By just getting the word processor started the buffers now contain bits and pieces of the **COMMAND.COM** and the **".EXE"** file of the word processor, possibly within the same buffer.

Lets compound the problem. You use the word processor to edit a letter to your girl friend which you started yesterday. For discussion purposes, lets say that your letter is exactly 400 characters in length including the end-of-file marker as the 400th byte. You make a few changes, save the letter back to its designated directory, and print the letter. You exit the word processor and return to the **DOS** prompt.

Next you decide to make a minor change to some control line in your **CONFIG.SYS** file. **DOS** loads the **CONFIG.SYS** file, all 100 bytes of it, into the same buffer used to control the writing of the 400 byte letter to its disk location. Lets further complicate the issue by saying that the disk is segmented into 1024 byte sectors. What do you think the **PRIVACY INVADER** has done to both the original letter and to the **CONFIG.SYS** file?

The 400 byte letter was written to a 1024 byte disk sector. **DOS** used the contents of two contiguous memory sectors to actually perform the physical write to the disk. The letter on disk is now trailed by 624 characters of residue which was in these two contiguous buffers at the time that **DOS** performed the actual disk-write command. Be assured that there are 624 bytes of appended garbage or something after the end-of-file marker because the disk must write 1024 bytes. There is no variable length write onto a hard-disk or diskette. It is almost like cutting up documents with scissors and pasting pieces of different documents together to make a new document.

Time also assists the **PRIVACY INVADER** in its contamination process. When the system was powered up, chances are that all the information that the buffers contained was code from the **COMMAND.COM** file. But the longer a system is used within the same application such as a word processor, the more the appended data problem or the **PRIVACY INVADER** threatens. The

reason for this is simple. Once enough of the buffers have been used for natural language (English, French, etc.) application, the more the contamination tends to be in natural language.

Likewise, while buffers are being used heavily to support graphic applications, the more the contamination tends to be in graphic language. This is logical since the use of all buffers are under **DOS** control.

While there are no rules established on just how it all happens, there is one rule to remember. The **PRIVACY INVADER** disease attacks 100 percent of all files with only one exception. Any file which is the exact length as the physical sectoring on disk escapes the **PRIVACY INVADER**.

The affect of the **PRIVACY INVADER** on any organization can be described in many scenarios but let's look at two. First the federal government has tons of privacy act information on unclassified **DOS** based systems. Without safeguards on these systems, privacy act data will become appended data by the **PRIVACY INVADER** and will unintentionally be given to others who have no need-to-know.

The **PRIVACY INVADER** guarantees the spread of its disease through the routine exchange of information by people using diskettes as the exchange medium.

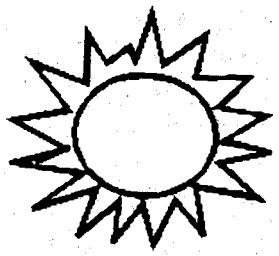
Likewise, in the classified world where systems process only one classification level, **DOS** will cause classified material from different documents to be "cut and pasted" together, violating the need-to-know principal and breaking classification security rules.

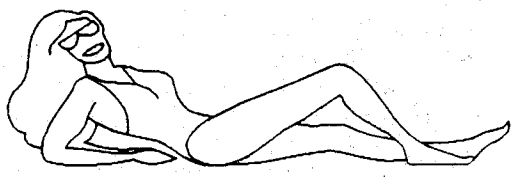
At this point, there is the question as to whether this disease should continue to be identified as the **PRIVACY INVADER** or a variant know as the dreaded **MISS CLASSED** disease?

MISS CLASSED

Worse than the **PRIVACY INVADER** disease on sensitive unclassified systems, the **DOS**-based **MISS CLASSED** disease occurs on all systems which process information of different classification levels. It is a very serious security threat. It is a variant of the **PRIVACY INVADER** in that it contaminates in a similar manner.

However, in order to be categorized as the **MISS CLASSED** disease, it must concatenate information pieces, two or more, from differing classification levels together.





Hypothetically, lets say that System A processes **UNCLASSIFIED** and **SECRET** level data. It is simple using the **DOS COPY** command to create a disk file containing **UNCLASSIFIED** data followed by **SECRET** data or a **SECRET** file appended with **TOP SECRET** residue.

And when the appended data is sent by unprotected mail or courier to an unauthorized environment, we unknowingly violate security.

The **MISS CLASSED** disease is the infection which contaminated the career of the fifteen year military officer mentioned earlier.

A picture is worth a thousand words so lets demonstrate the problem--just how does the **PRIVACY INVADER** and **MISS CLASSED** diseases do their dirty work? If you are not computer literate, get someone to do the following demonstration on a **DOS** system with a hard drive.

Step One. Ensure that the system prompt points to the root directory. Use the **DOS TYPE** command -- type the **CONFIG.SYS** file. Get a mental picture of its contents or do a **PRINT SCREEN** to capture the actual contents. Remember that the **TYPE** command will only display the number-of-characters as recorded in the directory. It does not display anything after the number-of-characters and the true physical end-of-file based upon disk sectoring.

Step Two. Use a hex editor such as Norton Utilities to view the **CONFIG.SYS** file. While displaying the file in natural language, notice the appended data attached to the **CONFIG.SYS** file.

Compare the **PRINT SCREEN** version to see just where the appended data starts. Look at the screen close. Try to identify the appended data. Again, get a good mental picture of its content or do another **PRINT SCREEN**. When you are finished looking, exit the hex editor.

Step Three. At the **DOS** prompt, perform a **COPY** or the **CONFIG.SYS** to an arbitrary file named **XXX.XXX**. This creates a new file using the same method which we use to transfer a file from hard disk to floppy disk. Reenter the hex editor and display the **XXX.XXX** file. Look at the change in the appended data. It will be different than that appended to the **CONFIG.SYS** file.

There may be odd looking **ASCII** characters which may be part of some **".EXE"** file. There may be natural language words, sentences, etc. And chances are that there will be pieces of multiple files as indicated by multiple end-of-file markers on the screen. If you are in a classified environment, look close, you may see the **MISS CLASSED** disease right there on the screen!

You may continue to **COPY** files and to use the hex editor to display other files. Case-in-point, in normal **DOS** systems every file will be contaminated and you have seen how the **PRIVACY INVADER** and **MISS CLASSED** diseases spreads their infections.

ILLEGAL OCCUPANT

Every personal computer user has at some time or other deleted a file of something which they no longer desired to keep. The DELETE command is notorious for leading an individual to think that the information is gone, deleted from the disk--one of the most popular misconceptions, far from the truth! What really happens is that the directory index is changed to reflect that the file is now deleted by the simple changing of the first character of the name of the file in the index.

This logically saves much time in the **DOS** disk management concept, especially under the older central processing units. It is the simplest and fastest way possible to **DELETE** a file. It eliminates the need of going through the very slow and agonizing process of

overwriting the entire data file. The person that originally conceived this scheme needs to be complemented, because it represented forward thinking in-so-far as systems design but on the other hand it represents a major flaw in the security arena. What really happens is that the data file is left intact on the diskette or disk for further recovery until **DOS** reassigns and overwrites the physical disk space with some other file. This could be immediately or it could take some time depending upon the use of the computer.

Stories of the 1960-1970 time frame were circulated about the Soviets and their front organizations. The stories related to how they would buy up obsolete systems and their magnetic media from almost any company in the California silicone valley area.

The systems they sought were from progressive high-tech companies which were themselves trying to stay up with technology. High-tech companies needed more processing speed to do their work so they frequently replaced their entire computer systems with faster more efficient systems. And in order to be efficient, these newer and faster systems needed the support of faster disk drives with more storage capacity. Floppy disk drives have been upgraded from their original 320KB of storage to the high density of 2.88MB.

So as technology changed, companies changed their hardware and software just to keep up with technology and to become more efficient.

The magnetic media used on the older systems was often sold with the system. It was reported that front organizations would literally purchase all of these "old systems" from the leading high-tech companies.

Much of the reasoning was to acquire high-tech information by exploiting the magnetic media for valuable contents. Who knows, maybe the United States really didn't need a ban on technology transfer. The transfer was happening on the older obsolete systems by itself, we were selling our high technology information in simple "garage sales".

The **ILLEGAL OCCUPANT** disease is another variant strain of the **PRIVACY INVADER** in that it too is spawned by **DOS**. It is simply the residue left in the unallocated space on magnetic media due to the method **DOS** uses to delete (eliminate?) unwanted files of information. It is hazardous to the health of any organization which tries to preserve its information. And since it leaves a tale-tale (tell-tell) trail of clues for the special computer crime investigator, it can also be hazardous to the health of individuals using computers for illegal activities.

DAA BLESSING

As documented in the **"THE ACCREDITOR'S TOOLBOX"** published in the Proceedings of the 1991 Third Annual Canadian Computer Security Conference, the Designated Approving Authority (**DAA**) is the individual authorized to accredit a computer system and to issue the official "approval to operate".

Since the **DAA** accepts security responsibility for the operator of the **AIS** and "officially declares that a specified **AIS** or network will adequately protect information against compromise", it is therefore the **DAA's** responsibility to approve and disapprove the use of "shareware".

Approval is based upon the DAA's review of the shareware program with consideration to its value to operations versus its potential threat. To minimize

the threat, the DAA usually requires the original source code of the "shareware" so that it can be examined for evidence of malicious code, etc.

Once examined and approved for use, it is then assembled or compiled by government personnel and distributed to its users using the government developed object code. Each shareware package is reviewed on a case-by-case basis. The concern is that shareware programs are usually developed without regard to security rules and that such programs are a prime means for transporting malicous code such as Viruses and Trojan Horses. Therefore, the general rule is that it is illegal to use shareware unless it has been obtained and released by the government with approval of the **DAA**.

Shareware can also have a copyright requiring a payment of some size to its legal owner. However, many users do not want to hear such rules especially if it perceived that the use of the shareware helps them "get the job done".

Every year, individuals exchange thousands of shareware programs over thousands of electronic bulletin boards. Many of these work their way illegally onto systems which have already been approved to operate by the **DAA**. Not only could an organization be held libel for payment to its owner, but the risk of not adequately protecting information is a direct concern of the **DAA**. Without adequate review and approval of shareware, there are no assurances. Some do not share the view that shareware poses a major threat, and maybe they are correct. However, that judgement does not belong to the casual computer user. The authorization, proliferation, and use of shareware is the DAA BLESSING Disease.

CHEAP USER

The **CHEAP USER** Disease is a variant of the **DAA BLESSING** Disease in that it is spread by human action. The **DAA BLESSING** Disease affects shareware software from shareware vendors the same as the **CHEAP USER** Disease affects copyright software from commercial software vendors. The fifteen year career officer had a case of the **CHEAP USER** Disease. He simply was too cheap to purchase his own software. Owning a computer but stealing software is just like owning a car but stealing gasoline to make it work.

There have been several lawsuits against organizations for open copyright violations. One $12,000,000 suit was supposedly won against the United States government by three vendors who jointly

sued for violations within one military service. One might argue that stealing software is not a computer security issue and that may be correct. Never-the-less, it is a problem which has to be recognized and managed.

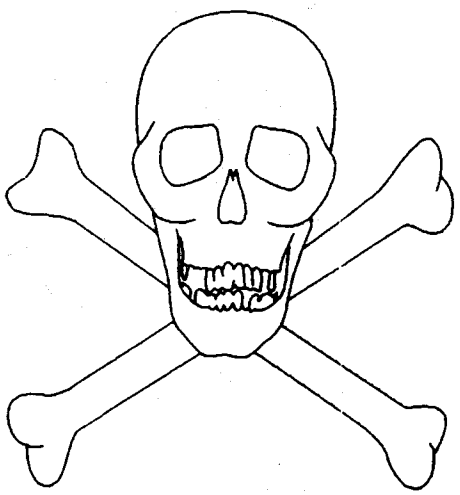
BIT DEATH

The last of our diseases is given the name **"BIT DEATH DISEASE"**. It is the result of those hundreds of computer viruses which are currently infecting millions of computers around the world and killing their operations.

In his article "The Kinetics of Computer Virus Replication" published in the Proceedings of the 1991 Third

Annual Canadian Computer Security Conference, Tippett uses a complex mathematical model to exptrapolate their reproduction characteristics. In summary, his analysis simply states that without effective measures to control the problem, viruses will redouble in number every 1.8 months.

Using his model, Tippett predicted that approximately 12 million of the 80 million computers worldwide could be infected just a short 48 months after the beginning of viruses in 1987.



This prediction would place a major threat to our systems now! In October 1989, International Business Machines listed 28 unique **DOS** viruses. This number grew to 250 by the end of 1990 and to 555 by the end of 1991. The number of unique viruses is exponentially growing.

At first I disagreed with Tippett's findings. But recently by simply observing the infection of systems within a relatively closed and benign environment such as the one where I work, I have changed my mind. I know that we are faced with a major problem and that we have to take strong measures to combat this major threat to our computer systems and their information.

For example, the **STONED VIRUS** has reached epidemic proportions on some United States Air Force bases.

And while the **STONED VIRUS** is more of a nuisance than it is destructive, it is still disruptive and is costing thousands of dollars just to remove it from those systems that have been infected. There are articles on the virus threat to computer systems in many newspapers and magazines. Therefore I don't need to expand on this subject. It is only necessary to ask -- Has the **BIT DEATH** Disease taken its toil on your organization, yet? What is the basis of your program to protect your systems? Is it adequate?

CONCLUSION

We have discussed six security related problems which exist in the day-to-day operations of personal computers. Again, they are:

(1) Appended Data within a file
(2) Classification Violations
(3) Residue in Unallocated Space
(4) Unauthorized Shareware
(5) Copyright Violations
(6) The Spread of Viruses

Programs have been developed to attack some of these problems, primarily in the virus arena. Pick up any PC magazine and you will find any number of anti-viral programs for sale by their vendors, some advertising a capability to detect 1000 viruses--a nice round number. One might think that the vendors themselves are propagatind viruses just to cash in on the action.

It is not easy to get people to understand the threat of these diseases to themselves, their job and their status. Likewise, the job of containing these six diseases is not easy because it is not easy to sell something that is seemingly not productive.

In a manual operation, it is not cost-effective nor is it a simple process to minimize these threats. It is extremely manhour intensive. Therefore, an

effective program to detect and minimize the threat of these diseases can only be done through a cost-effective time-reducing program utilizing automation.

The responsibility of looking for and detecting these day-to-day computer security diseases has to be the responsibility of one individual within each organization. By whatever name, that person is the Computer Security Officer. It is his/her job to manage the computer security of an organization and in doing so, to minimize the threat of all these diseases.

In the beginning I mentioned "The Computer Security Toolbox" which was developed by the United States Air Force Intelligence Command and that it was a vaccination tool against all six of these diseases. I will not document the toolbox and its content in this article since it contains some programs which must be protected against piracy. I also don't want the private contractors to get too rich by duplicating our work. However, a live demonstration will reveal its capabilities. I hope you enjoy the demonstration.

E-Mail Privacy and the Law

Christine Axsmith, Esq.
ManTech Strategic Associates

Introduction

A strong privacy policy protects the rights of employers and employees. Clarification and communication of that policy to employees protects against the uncertainty of the direction the law might take in the future on this issue.

In a classified environment, E-Mail monitoring is justified by concerns of waste, fraud, abuse, and espionage, any of which could lead to a criminal investigation. Privacy standards are determined by the Fourth Amendment to the Constitution, and are applicable to all Americans. The basis for these standards is the reasonable expectation of privacy test. That test is: 1) previous practices by the government employer, 2) written policies outlining the extent and manner that an employee can expect his or her privacy to be reduced, and 3) notice to the government employee of what privacy he or she can expect in the workplace. Privacy standards are also established by statute, and the ones applicable to federal employees are sometimes qualified by national security.

Other problems can complicate E-Mail monitoring, such as civil lawsuits by employees claiming and invasion of their privacy. The federal statutes discussed in this article are: The Privacy Act[1], and the Electronic Communications Privacy Act[2]. Common law tort is a basis for a civil suit as well. An effective way to handle privacy issues is to develop a thorough privacy policy which will spell out for the employee the amount of privacy that can be expected in their federal workplace. A privacy policy will give notice to employees that affect their expectation of privacy which will influence a court when it decides if the employee claiming the privacy invasion was reasonable.

Potential civil lawsuits will be reduced by informing workers of the privacy policy as worked out by management. In addition, information obtained as a result of E-Mail monitoring will be available for use in criminal investigations.

This article does not suggest only one approach to an E-Mail policy. It outlines considerations in determining an E-Mail policy, and makes several suggestions to help' federal agencies clarify their privacy policy regarding E-Mail, particularly in a classified environment.

This area of law is not settled, and the final answer depends in large part on future court decisions. Between now and then, privacy concerns will need to be addressed by your organization, and this article will discuss the background of the legal issues. Specifically, this article addresses electronic mail privacy in a classified environment. A conservative approach is taken in this article. Another term to use would be "cautious." Since the rules of law in this area are so unsettled, covering all of the angles is important to establish a policy that will withstand a court decision not necessarily sympathetic to national security interests. It is never far sighted to

1. 5 U.S.C. §552a
2. 18 U.S.C. §2510

assume that the courts will agree with a certain perspective consistently and exclusively.

Problems With Privacy

Alana Shoars was an E-Mail Administrator for Epson America. One day she came into work discovered her supervisor printing our E-Mail messages between other employees. In her position as E-Mail Administrator, she often assured other employees that the messages they sent were private, and that the privacy would be respected. Ms. Shoars was terminated a day after she questioned the practice by the company manager[3].

Within the last year or so, employees have begun to sue their employer for reading the E-Mail messages of its employees[4]. Since that time, many complicated questions have been raised about the right of employers to read E-Mail resident on corporate computers. The issue has primarily been raised in wrongful termination claims. In California, a recent change in the law has made the standard wrongful termination claim less attractive to litigators, or at least less profitable. Damages for wrongful termination claims are now limited to lost wages[5]. Claiming an invasion of privacy can allow the employee to claim damages other than lost wages.

None of these lawsuits involve federal employees. None of them involve a classified working environment. Nevertheless, the ramifications of recent developments in litigation are too serious to ignore. Will employees have the right to sue their employer for monitoring their E-Mail messages? Does that in turn mean all E-Mail monitoring could be effectively eliminated in the future? What is the current status of the law on this issue?

Serious questions need to be raised and addressed in privacy policies of government employers. Legally, questions on E-Mail privacy have not been addressed by the courts. Often, years go by before a definitive answer to specific issues, such as E-Mail monitoring in a classified environment, would be addressed. Before then, decisions must be made on what a privacy policy on E-Mail will include and what it will not.

Background

Privacy as a legal issue can be divided into two parts: civil and criminal. As a civil issue, a lawsuit could be based on state or federal law. However, state law does not apply to a federal workplace. The cases filed against Epson America and Nissan Motor Company are based on California law, which cannot be applied to a claim by federal employees. There, discharged employees are claiming an invasion of privacy because they were terminated based on the content of the employees' E-Mail messages.

Criminal Law

As a criminal issue, reasonable expectation of privacy involves the Fourth Amendment and the warrant requirement for searches and arrests. If a person's reason-

3. NY Times, Dec 8, 1991; Section 3, "Do Employees Have a Right to Electronic Privacy", by Glen Rifkin

4. National Law Journal, Sept 16, 1992; "The Outer Limits", by Rosalind Resnick

5. LA Times, Oct 26, 1991 Saturday; Home Edition Part A, Page 1, Column 1; "Job Loss Suits Take a New Twist", by Terry Pristin

able expectation of privacy is violated, the evidence found as the result of that invasion into privacy, cannot be used as a basis for an arrest.

Privacy encompasses criminal and constitutional aspects regarding the Fourth Amendment to the Constitution of the United States. Specifically, they relate to search and seizure of evidence, and arrests pursuant to a warrant. This article does not address the issues surrounding admissibility of E-Mail messages at a trial as evidence. Even if the information discovered during E-Mail monitoring is not admissible as evidence at trial, E-Mail most likely could be used as a basis for an arrest or search warrant if the suggestions of this article are followed. Results of E-Mail monitoring can be used in future criminal action if illegal activity is discovered. A final legal answer about the use of information obtained during E-Mail monitoring has not been given by the courts.

The question here hinges on the reasonable expectation of privacy of the individual who is the subject of the search. "Computers" as such, are not the issue. The issue is the reasonable expectation of privacy of a federal employee at work under the Fourth Amendment standards created by the courts. Reasonable expectation of privacy protects people not just places[6]. What constitutes a reasonable expectation of privacy for E-Mail users in a national security environment is still undecided. One day a clear definition may exist, but no one today can foresee a Supreme Court decision made tomorrow.

If the reasonable expectation of privacy of the individual whose E-Mail is being searched is violated, the evidence obtained as a result of the privacy invasion could be excluded at trial, or an arrest based on that information could be invalidated. If a person's reasonable expectation of privacy is violated, the intrusion becomes a search and without a warrant, the information discovered by the intrusion cannot be used as a basis for an arrest warrant. So the concern becomes one of ensuring that whatever information is gained can be used as an adequate basis for a search or an arrest warrant. Currently, courts recognize a reduced expectation of privacy for government employees in the workplace. The person must have a sincere expectation of privacy and that belief must be reasonable in our society for the courts to recognize that a "reasonable expectation of privacy" exists in a legal sense. The issue of a public employee's reasonable expectation of privacy was addressed by the courts in the seminal case of U.S. v. SPEIGHTS[7]. In that case, a police officer stored a sawed-off shotgun in his locker. In the course of investigating a breaking and entering ring, Speights' superiors received information that he stored an unregistered weapon in his locker. Eight lockers were searched, including Speights'. His locker was secured by a personal lock and a police-issued lock. Speights' locker was opened without a warrant. There was no regulation on the use of private locks, and no regulation or notice to the ranks that the lockers might be searched.

The court found that Speights had a sincere expectation of privacy in his police locker. To decide whether the government employee's expectation of privacy was reasonable, the court weighed several factors. The factors the courts rely on to determine if the expectation of privacy was reasonable are: 1) previous practices by the government-employer, 2) written policies outlining the extent and manner that an employee can expect his or her privacy to be reduced, and 3) notice to the government employee of what privacy he or she can expect in the workplace. The rules on reasonable expectation of privacy in the workplace for federal employees

6. Katz v. United States, 389 U.S. 347 (1967)
7. United States v. Speights, 557 F. 2d 362 (1977)

differ from and are less stringent than the rules on reasonable expectation of privacy for private sector employees. The practical effect of this difference is that many of the cases now publicized involve private sector employees and do not use the same rules that would be used in a case involving a federal employee, especially in a classified environment.

In a national security environment, the reasonable expectation of privacy for employees is reduced even further than for regular government employees. But since the law in the area of E-Mail privacy is far from settled, clarifying the situation to the employees is important. In the past courts have weighed heavily the employee's subjective idea of reasonable expectation of privacy when the policies of the government-employer have not been clarified. Merely because the intrusion into privacy is possible (e.g. that the system manager can read all messages on the system) does not necessarily reduce the reasonableness of the employee's expectation of privacy.

The ability of the system manager to read everything on the system does not mean the user of the System has no reasonable expectation of privacy. To a system manager, or to anyone familiar with computer systems this interpretation may seem illogical. If a person has access to the entire system, then it would seem that any privacy expectations on the part of a user would be unfounded.

However, the definition of "reasonable" that counts is the definition adopted by the courts, who will probably consider the perspective of the user, at least in part.

Certain definitions of "reasonable expectation of privacy" have been attempted regarding E-Mail privacy, but none of them apply to a federal government computer system. The ruling in Speights is the current legal test used to determine the reasonable expectation of privacy held by a federal employee at work.

The Electronic Communication Privacy Act[8] also establishes criminal sanctions for interception of electronic communication. The statute calls for imprisonment of not more than five years, a fine, or both. An exception is granted for someone acting under "color of law" if one of the parties to the communication has given prior consent. Another exception is where the one intercepting the message is a party to the communication, or one of the parties to the communication has given consent to such interception, which does not apply if the purpose of the interception was commit a criminal or tortious act.

Civil Law

While not applicable to a federal privacy claim, the legal trend in California is interesting because similar claims may very likely be filed in other state jurisdictions, and possibly in federal court in the future.

Federal law governs in a federal workplace. Several federal laws address privacy, such as the Privacy Act[9]. The Privacy Act concerns information that the federal government gathers and keeps concerning individuals, but in the language of the act, the right to privacy is qualified by national security concerns. The Privacy Act also limits its scope to "a system of records," and its applicability to an E-Mail system is unsettled. Even if it did apply, however, the Privacy Act also excepts "files the disclosure of which would clearly constitute a clearly unwarranted invasion of privacy." The Privacy Act itself is part of a larger piece of legislation called the

8. 18 U.S.C. §2510
9. 5 U.S.C. §552(a)

Administrative Procedure Act[10]. The Privacy Act specifies what information should be disclosed to the public, and how that should be done to protect individual privacy. Nevertheless, neither exception has been irrefutably applied to the content of government employee's E-Mail. As long as the courts see that some right to privacy exists, there is a basis for a civil privacy lawsuit.

Electronic Communications Privacy Act extends existing privacy protection for oral and wire communications to electronic communications. Under the ECPA, only if the sender or recipient of an electronic message gives permission, and the computer system allows access to a computer outside the corporate computer, can an employer read the employee's E-Mail. An exception to the privacy guarantees of the ECPA exists for conspiratorial activities that threaten the national security. So there is a national security exception to this statutorily defined right to privacy as well.

Lawsuits will arise in the future on the issue of E-Mail privacy, giving rise to potentially large legal expenses. Notice to users of their privacy rights will make the users aware of their situation and will hopefully conduct themselves in accordance with that knowledge. Also, "notice" will reduce the reasonableness of a user's subjective expectation of privacy, when prior notice was given to the opposite effect. The likelihood of a successful lawsuit by federal employees for E-Mail monitoring, using the ECPA is very slight because of the national security exception written into the language of the ECPA.

Common law tort is another basis for a civil suit for the federal employee. The legal standard used is the same for this cause of action as it is for "reasonable expectation of privacy" in a criminal context. The same factors from SPEIGHTS apply, i.e. notice, policies of the government employer, and the practices of the government employer. A court will balance these three factors, deciding whether the employee had a sincere expectation of privacy, then whether that expectation was reasonable in our society. If the answer to both issues is yes, then the employee will win the suit.

Suggestions

Most legal departments can prepare language for a privacy notice declaring there is no reasonable expectation of privacy on the computer system. Until the courts interpret that language to mean precisely that regarding E-Mail privacy for government employees, the viability of such a solution is doubtful because the courts most likely will weigh the written policies of the government employer and previous practices. Notice adequately addresses the notice requirement to reduce the reasonable expectation of privacy of the employee-users. However, the courts may decide to balance several factors when the issue is decided, of which notice may be only one.

For a conservative approach to help ensure that a privacy policy will stand in the face of future legal decisions, a strongly worded notice should be combined with policies reflecting the approach taken by a particular agency to the privacy in its E-Mail system. These policies should be clear and thorough. Following through on these policies should be the actions of the agency.

Department policies should outline E-Mail auditing, how often it will be done, who can expect to have their E-Mail read, and who has the authority to read the E-Mail of others. Strongly worded notice to the user delineating the extent of privacy on a computer system is only one factor a court could use in deciding whether or not perusing the E-Mail of others is permissible in a classified environment. But the

10. U.S.C. §§551 et seq.

effect of the notice would be strengthened by clarification and communication to users through department policies and practices. The legal standard to date depends in large part on the policy of the government employer, and how that policy is communicated to the employee whose privacy is being reduced. A policy of reduced expectation of privacy can be further strengthened by a written statement of who in the organization has the right to review other people's E-Mail and who does not.

This article suggests a conservative approach in protecting national security interests in the long run. The aim of these suggestions is to help ensure that information gathered through the monitoring of E-Mail will be available as a basis for an arrest or search warrant when the issue is decided by the courts, and to forestall potentially large legal expenses in the future from civil lawsuits.

Clarification and communication of a privacy policy by notice to the user, written policies, and practices will prevent misunderstandings and lawsuits in the long run.

ELECTRONIC MEASUREMENT OF SOFTWARE SHARING FOR COMPUTER VIRUS EPIDEMIOLOGY

Larry de La Beaujardiere
Department of Computer Science
University of California
Santa Barbara, California 93106

Abstract

Mathematical models in computer virus epidemiology employ simplified assumptions about how software is shared among groups of users. These models would benefit from accurate data describing actual software sharing patterns. An algorithm is presented which records the topology of the software interchange network. The algorithm proceeds as an "epidemic" computation, and is forwarded by users along with a replicated database when they share software.

Introduction

The word *virus* to describe unwanted computer code was first used by novelist David Gerrold in *When Harley Was One*, in 1972 [1]. The term was formally defined by Fred Cohen in 1983. Informally, it is a computer program that alters other programs to include a possibly modified version of itself [2].

Viruses pose a significant threat to the computing community, especially given the continuing proliferation of personal computers and associated software applications. As of 1989, some 76 distinct viruses which operate on a variety of personal computers had been identified. Dozens of variations of these viruses were also identified. These viruses threaten the integrity of users' files and programs, and affect the availability of computing resources. Furthermore, considerable time and money is spent by individuals in preventing and recovering from viruses.

In principal, a computer virus can spread to the transitive closure of information flow [2]. The techniques of mathematical epidemiology, as traditionally applied to the study of biological disease, have been used to study this spread. However, epidemiological models suffer from the effects of simplifying assumptions about the patterns of software sharing among users. There remain doubts as to the accuracy of these assumptions; as a result, the research community has called for formal study of the software interchange process.

The procedure presented in this paper would produce accurate and detailed measurements of the topology of software sharing. Our proposed method depends upon the active participation of the user involved in exchanging software. It is administered by some agency or authority, which is responsible for initiating and distributing the procedure, and collecting its results upon termination.

The algorithm functions as an *epidemic computation*, much like viruses themselves. Users who share software via removable media are instructed (requested) to also include the proposed program and an associated data file. The data file is an element of a replicated database, and contains a representation of a directed subgraph that is a component of the software sharing network. The recipient of the shared software runs the algorithm on his machine, and thereby includes himself in the network.

If the transported algorithm finds that it is making the first visit to a given machine, it generates a name for the machine and initializes a local copy of the data file. The local data file will then represent a graph with one node: the destination machine. The algorithm merges the graph that was transported from the source node with the existing graph on the destination node. In this manner, the algorithm and database are distributed throughout the population, and a picture is built of the topology of software sharing.

At some point (perhaps in response to size constraints), the algorithm terminates. The many results are sent by cooperating users to the administering agency, to be digested, reconciled, and analyzed by virus epidemiologists.

For ease of exposition, and to enhance user acceptance of the procedure, a catchy name has been adopted: SWIMS—the SoftWare Interchange Measurement System.

In subsequent sections, we present the motivation behind a proposal of this kind. Current epidemiological models are presented, and we discuss how these models could benefit from SWIMS. The procedure is explained, and other epidemic algorithms are mentioned.

The implementation of SWIMS presents specific challenges. These are discussed, and a modest approach is suggested to initially test the concept among a small population. SWIMS is geared towards sharing via removable media; the measurement of other exchange modes is discussed.

Motivation

Since the computer science community is likely to view our proposal as controversial, some motivating factors are discussed.

It is axiomatic among computer security specialists that awareness of threats is the first step in ensuring a secure environment. The SWIMS procedure will involve the general population of computer users in the process of understanding computer viruses. With features to graphically display the evolving view of the interchange network, SWIMS will foster a strong consciousness of the far-flung community of users in which we are members.

There exist many approaches to virus defense at the level of the individual computer [1,3,4]. However, Cohen has shown that prevention of infection can only be guaranteed

by limiting information flow or functionality. Further, he has demonstrated that, in general, the problem of virus detection is undecidable [2]. Thus, the partially effective response at the level of the individual should be supplemented with a response at the global level. Accurate epidemiological models can help formulate a global response—the proposed algorithm could serve as its backbone.

The most obvious way to collect data about software sharing is through printed surveys distributed to a group of users. The problem with such a survey is that it would be impossible to integrate the results into a comprehensive view of the topology of sharing. Individuals could describe how often and how distantly they share, but one could not get a sense of the width and connectivity of localized sharing groups. The connectivity between sharing clusters, and their intersection, would not be evident in the data.

Epidemiological Models

The goal of modeling in computer virus epidemiology is to gain insight into the large-scale behavior of viruses. This includes an understanding of what fraction of the population is infected at a given time, the duration of the epidemic, the spatial distribution of infected individuals, and the probability that a given individual is infected.

The population under study is divided into classes of individuals (*ie,* computers), according to whether they are susceptible to infection, infected, or removed (cured) [5]. In the simplest models, denoted SIS (susceptible→infected→susceptible), the removed state does not appear. A more likely model for the computer domain is SIR (susceptible→infected→removed), in which users who have been infected by viruses adopt protection procedures and anti-viral software. The true picture probably lies somewhere between the extremes presented by these two approaches [6].

An epidemiological model attempts to describe the transmission process. Most of the existing work in computer virus epidemiology is based upon a homogeneous mode of transmission [6]. This means that any individual is equally likely to infect any other individual in the population. Software sharing is actually a heterogeneous process, since a user typically has a group of contacts with whom she frequently shares. She exchanges outside this group relatively infrequently.

Thus, some spatial or logical structure is best applied to the model, to capture the notion of *proximity* or *locality* between users (proximity refers to the likelihood that two individuals will share software, and does not necessarily imply physical propinquity).

The most comprehensive work along these lines is by Kephart and White, of the IBM T. J. Watson Research Center [6]. One of their models is based on a directed graph, in which each vertex represents an individual computer, and all of the $N(N-1)$ possible edges have the same probability of being included in the graph. The edges and vertices are associated with an infection rate and a cure rate, respectively. Although this model is heterogeneous, it does not model well the proximity relationship, because every possible edge has an equal chance of inclusion.

To capture the effect of proximity, Kephart and White present a hierarchical model

in which the population is logically organized into a tree structure. The individuals are represented at the leaves, and the probability that one individual infects another is a function of the number of levels that must be climbed to reach a common ancestor.

The authors make simplifying assumptions which affect the degree to which the hierarchical model reflects actual sharing patterns. The tree is assumed to be binary, and the rate of infection between nodes decreases geometrically as they are increasingly removed one from another.

A cellular automata model is also employed. It is highly oriented towards assessing the effects of locality in sharing. The susceptible neighbors of a given individual are located in a square block of cells centered around the individual, and a uniform infection rate is applied to all cells in the block.

The Need for Refinement

Kephart and White have provided an important foundation with these models. However, they have pointed to a need for the subsequent development of their complexity. They state that the results obtained from their graphical and cellular models represent the extremes of the effect of locality in program sharing, and that the actual situation may lie somewhere in the middle. They have called for more research into actual sharing habits, and for a centralized authority to collect virus reports. The SWIMS procedure presented in this paper addresses these needs.

The directed graph model uses a uniform distribution of edges. However, sharing probably involves local well connected clumps of users, with fewer connections between users in different clumps. SWIMS data could be used to determine the appropriate non-uniform distribution of edges in the graph.

For the hierarchical model, the SWIMS results could be used to determine the branching factor of the tree as a function of tree level. An improved relation between infection rate and tree level could also be derived.

The cellular automata model could be refined by imposing a non-uniform distribution of the infection rate over an individual's neighborhood. Also, neighborhoods could be constructed with varying shape and size.

However, without real-world data on the program interchange network, specifying these improvements to the model is dangerous speculation.

The SWIMS Procedure

SWIMS is billed as a *procedure* rather than an *algorithm* because it relies on user participation. Another participant is the central agency that initiates the procedure and collects the results. Since the goal is to measure software sharing, we directly associate the procedure with the act of sharing: when a user shares software with another, he also shares the SWIMS program and associated data file.

Once at his machine, the recipient runs SWIMS. If his machine has never been exposed

to SWIMS, the program will copy itself onto his hard disk and initialize the required data files. Thus, a database containing one node is created. SWIMS will then merge the graph that came from the source machine with the graph at the destination machine, to update the combined picture.

We introduce the term *epidemic computation* to describe the operation of SWIMS, since it involves attachment to host individuals and replication throughout the population. Thus, it resembles an actual computer virus, with the important difference that replication occurs at the mutual discretion of both users involved. It differs from the usual notion of a distributed computation in that the message passing channels are not predefined.

The three steps of SWIMS are summarized:

- **Initiation:** This takes place at the launch sites chosen by the administering agency, or when the algorithm propagates to a new node. SWIMS depends on unique identifiers for each node. This is best accomplished by letting the user choose a 20-character alias for himself. Uniqueness of id's can be ensured by concatenating the alias with the current system time.

- **Propagation:** At the source of software sharing, the program and database are copied by the user onto removable media. At the destination, a merge on the two directed graphs is performed, with nodes in common identified, and edges placed accordingly. The resulting graph serves as the updated database for the destination site.

- **Termination:** The procedure will automatically notify the user that it should terminate in response to some predefined condition. This could be that the data file has exceeded a specified size, or that the system clock has advanced past a termination date. At this point, users mail their data file on a diskette to the central agency, with the proviso that the agency will return the diskette. A reward incentive could be employed to maximize cooperation.

Challenges Posed by SWIMS

Tampering

Because of its wide distribution, SWIMS would be a likely carrier for a virus attack. This problem could be mitigated by employing some standard virus defense techniques [2]. The algorithm could be offered as source code in a variety of popular languages. Since the bulk of the computation is a straightforward graph merge operation, the code would be relatively short and amenable to quick inspection.

Another approach would be to document a calculation of the checksum of the program, to be distributed on printed matter. Users could verify the program by comparing the expected and observed checksums.

Finally, SWIMS could be run in conjunction with a modified command interpreter that requires explicit authorization for any modification of data objects. This is an imperfect method for virus defense in general, since it generates too many "false positives" of virus

detection. The user is soon conditioned to authorize all changes [4]. However, when used with only a few programs (*eg*, SWIMS), this method can help defeat viruses.

Another weakness is the potential for sabotage of the SWIMS data file. At worst, this would create a set of subgraphs that incorrectly reflect sharing patterns. Statistical techniques could be applied to characterize SWIMS data upon collection, and outliers could be discarded.

These dangers will be minimized if the procedure is introduced in a smaller, more cohesive community than the general population, such as a University or a small town in the Midwest.

Big Brother

Americans are famous for resisting attempts by any agency to accumulate and centralize information about the population. Their fears should be lessened by the fact that users of SWIMS are permitted to identify themselves in the replicated database, and can therefore choose aliases. With a proper public relations campaign, the computer community could be encouraged to take part in this great National experiment.

Size Complexity

The size of the data file is an issue, since we can expect SWIMS to collect thousands of nodes in its graph. Cohen has claimed that a typical PC-based virus can spread to thousands of computers in a matter of weeks [4]. The storage requirements for the graph is $O(E)$, where E is the number of edges. In a complete directed graph, this is $O(N^2)$. However, the density of edges in the software sharing graph is likely to be sparse, bringing the size closer to $O(N)$. The ideal representation of the graph on disk is as a list of edges, organized by the node from which they are exitant. Unbounded growth of the data file is checked by algorithm termination.

A Vision for the Future

All of the foregoing challenges, if left unaddressed, conspire to diminish user acceptance of SWIMS. However, SWIMS appears more palatable in conjunction with a revolutionary vision of personal computing for the future. In this vision, the typical user is more savvy about threats to computer security. SWIMS helps to increase his awareness. The future user is accustomed to being involved in electronic networks. As on-line data connections into the home are increasingly utilized for mail and news delivery, SWIMS acts as a similar network for standalone computers. The proposed epidemic computation can contribute beyond providing data for epidemiological research. Given widespread use, it could serve as the basis for a protocol which tracks the source and progress of viruses. It could identify and warn individuals who are at high risk from a particular virus attack.

Other Software Exchange Modes

SWIMS is meant to measure software interchange via such removable media as diskette and cartridge tape. More direct means of sharing should also be measured in support of research in virus epidemiology. These sharing modes are more easily analyzed than sharing by floppy.

It is not clear to what extent software obtained from bulletin boards and news networks contributes to viral spread. Although these transmission means are always mentioned by virus experts, Cohen has asserted that only one widely known virus has ever been launched through a bulletin board [4]. Since this transmission mode involves on-line data access, and possibly subscription to a paid service, it would be feasible to establish an auditing procedure to directly measure frequency and location of use.

Viruses have also been known to spread via mass distribution. The MacMag virus was shipped with about 10,000 shrink-wrapped copies of Aldus Corporation "Freehand," [7][4]. In 1989, the AIDS virus was shipped to tens of thousands of users on a PC mailing list[1] [4]. This kind of virus attack is anomalous. It should not be incorporated into epidemiological study.

Other Epidemic Computations

SWIMS is not the first algorithm that proceeds as an epidemic through a system of separate computers. Researchers at the Xerox Palo Alto Research Center were the first to describe worm programs [1]. They used a worm to perform hardware diagnostics of Ethernet interfaces on a network of workstations [8]. They refer to their software system as *programs which span machine boundaries.* Other investigators at Xerox PARC have used randomized algorithms to distribute updates of a replicated file. The updates spread as an epidemic with a limited lifetime; infection ceases when too many attempts are made to reinfect updated nodes [9].

As early as 1964, an epidemiological model was applied to the transmission of ideas. The method was intended to help determine when an information retrieval system should be introduced as a tool to a population of scientists [10].

The existence of these other epidemic computations should help justify SWIMS to the personal computing community.

Conclusions

We have described the need for additional research into computer virus epidemiology, and demonstrated that current models of program sharing suffer from oversimplifications. Our proposed procedure, the SoftWare Interchange Measurement System, responds to these needs by providing comprehensive data on software exchange. To ease the accep-

[1] [4] and [1] contain contradictory descriptions of the AIDS virus.

tance of this controversial proposal, we have presented a number of ways to deal with the challenges it poses.

An attempt has been made to incorporate SWIMS into a vision of the future of personal computing—a future characterized by a knowledgeable user who is cognizant of her presence in a global computing network.

References

[1] E. H. Spafford, K. A. Heaphy, D. J. Ferbrache, "A Computer Virus Primer," *Computers Under Attack: Intruders, Worms, and Viruses*, P. J. Denning, ed., Addison-Wesley Publishing Company, Reading, MA 1990.

[2] F. Cohen, "Models of Practical Defenses Against Computer Viruses," *Computers & Security*, Vol 8, 1989.

[3] M. H. Brothers, "Computer Virus Protection Procedures," *Computers Under Attack*, 1990.

[4] F. Cohen, "Implications of Computer Viruses and Current Methods of Defense," *Computers Under Attack*, 1990.

[5] H. A. Lauwerier, *Mathematical Models of Epidemics*, Mathematisch Centrum, Amsterdam 1981.

[6] J. O. Kephart and S. R. White, "Directed-Graph Epidemiological Models of Computer Viruses," *1991 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1991.

[7] H. J. Highland, "Computer Viruses—a Post Mortem", *Computers Under Attack*, 1990.

[8] J. F. Shoch and J. A. Hupp, "The 'Worm' Programs—Early Experience with a Distributed Computation," *Computers Under Attack*, 1990.

[9] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic Algorithms for Replicated Database Maintenance," *Operating Systems Review*, Vol. 22, 1988.

[10] W. Goffman and V. A. Newill, "Generalization of Epidemic Theory: an Application to the Transmission of Ideas," *Nature*, Vol. 204, October 1964.

ENFORCING ENTITY AND REFERENTIAL INTEGRITY IN MULTILEVEL SECURE DATABASES†

Vinti M. Doshi and Sushil Jajodia‡

The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102-3481

ABSTRACT

Entity integrity and referential integrity are two important integrity constraints in relational databases. This paper first defines these integrity constraints in the context of single-level relations, followed by their extension to multilevel relations. While the concept of entity integrity extends to multilevel relations in a straightforward way, various referential integrity rules often create a conflict between secrecy and integrity requirements. Different referential integrity rules have been investigated with secrecy and integrity in mind, since both are important to multilevel secure databases.

1. INTRODUCTION

Secrecy and integrity are two of the most frequently heard concepts in the database world. Secrecy refers to the protection or safety of the data against unauthorized disclosure, and integrity refers to the correctness or accuracy of data. Preserving the accuracy of information is extremely important in any database. In the relational model, preserving accuracy of information is preventive in nature and is achieved by use of integrity constraints. Entity integrity and referential integrity are the two of the most important integrity constraints. They apply to all relations and should be enforced by the database management system (DBMS).

Although some aspects of referential integrity have been examined in the context of secure multilevel databases by many researchers [BURN88, GAJN88, LUNT90], there is still a need for a thorough and complete analysis of the various referential integrity rules with respect to multilevel secure databases. The purpose of this paper is to try to fulfill this need.[1]

The organization of the paper is as follows. Section 2 of this paper briefly reviews entity and referential integrity in single-level relational databases. In section 3 we first extend these basic concepts of referential integrity to multilevel relations. We then discuss the basic requirements to provide referential integrity control in multilevel secure databases under different levels of security labeling.

2. ENTITY AND REFERENTIAL INTEGRITY IN SINGLE-LEVEL DATABASES

Entity integrity and referential integrity are two basic integrity requirements in a relational database. They prevent incorrect data from being entered into the database. Entity integrity guarantees a unique representation of each entity in the database through specification of primary key attributes for each relation. Referential integrity assures that if there exist any references between two or more entities, then the related entities do exist in the database. Referential integrity is an inter-relation integrity constraint and is achieved with the use of foreign key attributes.

---

† This work has been supported by the National Computer Security Center, contract number DAAB07-91-C-N751.

‡ Also affiliated with the Center for Secure Information Systems and the Department of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030-4444.

1 We refer the reader to [DOSH91] for a more detailed exposition.

Entity integrity is defined as follows: A tuple in a relation cannot have a null value for any of the primary key attributes.

Before we can define referential integrity, it is necessary to define the concept of a foreign key. The definition of a foreign key requires two relation schemas: a *referencing* relation and a *referenced* relation. Let **R1** and **R2** be two relation schemas, and let R1 and R2 denote the relations corresponding to **R1** and **R2**, respectively. Let PK denote the primary key of **R2**, and let FK denote one or more attributes of the relational schema **R1**. FK is said to be a *foreign key* of **R1** if given any tuple t1 in R1, the following two requirements are met:

1. t1[FK] either does not contain any null values or contain only null values, and
2. whenever t1[FK] is non-null, there is a tuple t2 in R2 such that t1[FK] = t2[PK]. Here R1 is the *referencing* relation and R2 is the *referenced* relation. Sometimes PK is referred to as the target value of the foreign key FK.

It follows from the definition of the foreign key that there is an identical matching primary key value in the referenced relation for every foreign key value in the referencing relation. It is important to maintain the integrity between the referencing values (foreign key values) and the referenced values (primary key values). This integrity constraint is called *referential integrity*. It ensures that the database does not contain any invalid or unmatched non-null foreign key values (i.e., those values that do not have matching primary key values in the referenced relation). Referential integrity for relations ensures that each non-null foreign key value matches some corresponding primary key value.

2.1 REFERENTIAL INTEGRITY RULES

Whenever two or more relations are related through referential constraints, it is necessary that references be kept consistent in the face of insertions, deletions, and updates to these relations. Date [DATE90] (see also [DOSH91]) identifies the following four rules to maintain consistency of references. Exactly which rule is chosen for a particular relation depends on the behavior desired by the underlying application.

1. Nulls Rule
2. Delete Rule
   a. RESTRICTED-delete
   b. CASCADES-delete
   c. NULLIFIES-delete
3. Update Rule
   a. RESTRICTED-update
   b. CASCADES-update
   c. NULLIFIES-update
4. Insert Rule

The rules above are not exhaustive. There can be additional options like conversation with the end user, transferring information to some other files, etc.

3. MULTILEVEL RELATIONAL DATA MODEL

There are four different ways of assigning access classes to data stored in relations. One can assign access classes to entire relations (*relation-level granularity*), to individual tuples (rows) of a relation (*tuple-level granularity*), to individual attributes (columns) of a relation (*attribute-level granularity*), or to individual elements of a relation (*element-level granularity*). In the definitions below, the most general case is considered, in which access classes are assigned to individual data elements stored in relations. The modifications required for relations at other levels of granularity are straightforward.

We view a multilevel relation schema as $R(A_1,C_1, A_2, C_2, \ldots, A_n, C_n, TC)$ .where each A_i is a data attribute over some domain D_i, each C_i is the classification attribute of A_i, and TC denotes the access class of the entire tuple. The domain of C_i is some set consisting of access classes, and the domain of TC is the sublattice containing the union of the domains of C_i, $i = 1, \ldots, n$.

For a multilevel logical relation schema R, there is one physical relation R_c per access class in the security lattice. A user having a clearance at an access class c sees the relation R_c which contains data at access class c or below. More formally a relation at an access class c has the form $R_c(A_1,C_1, A_2, C_2, \ldots, A_n, C_n, TC)$. R_c consists of a set of tuples of the form $(a_1, c_1, a_2, c_2, \ldots, a_n, c_n, tc)$ where each a_i lies in the domain D_i or a_i is null, $c \geq c_i$, and tc is the least upper bound of all c_i, $i = 1, \ldots, n$. The access classes of the primary key values are always identical. If a_i is null, its classification attribute ci is identical to the access class of the attributes constituting the primary key.

3.1 POLYINSTANTIATION IN MULTILEVEL RELATIONS

The notion of a primary key is a fundamental concept in the world of single-level relational databases. Unfortunately it does not extend straight-forwardly to the multilevel world. This is because the *-property, which does not allow any write downs, must be preserved, and signalling channels must be avoided. Problems arise in multilevel relations when a user tries to enter another tuple with the same primary key value as that of an existing tuple at another access class. This update cannot be allowed if a relation's entity integrity is to be preserved. On the other hand, if the user is not allowed to insert the tuple, then either there exists a signalling channel (if the user is low) or the database suffers from a denial of service (if the user is high).

These security considerations have led to the notion of *polyinstantiation* [DENN87] (see also [JAJO90, JAJO91a, JAJO91b, LUNT90]). Polyinstantiation forces a relation to contain multiple tuples with the same primary key distinguishable by their classification levels or by the non-primary key attributes of the relation. The debate continues as to whether polyinstantiation is needed in multilevel relations or not. If polyinstantiation is not required in multilevel relations, then there has to be a solution to close signalling channels. If polyinstantiation is made a requirement for multilevel relations, then the question is how polyinstantiation should be managed.

A thorough discussion of all the issues associated with polyinstantiation is beyond the scope of this paper. For the purpose of this document, the discussion of polyinstantiation will be confined to the impact it has on referential integrity. In the remainder of this document, it will be assumed that there is a user-specified primary key consisting of a subset of the data attributes and not including the security classification [DENN87]. It will be called the *apparent primary key* of the multilevel relation schema.

3.2 ENTITY INTEGRITY IN MULTILEVEL RELATIONS

We assume that there is a relation schema R with a user defined apparent primary key, consisting of one or more data attributes of R. It will be denoted by PK. The entity integrity property of the standard relational model can be extended to the multilevel environment by defining the following three requirements (see, for example, [DOSH91] for a justification of these requirements).

A multilevel relation schema R is said to satisfy *entity integrity* if for all relation instances R_c of R and, tuple $t \in R_c$,

1. if $A_i \in$ PK then $t[A_i] \neq$ null,
2. if $A_i, A_j \in$ PK then $t[C_i] = t[C_j]$, i.e., PK is said to be uniformly classified, and
3. if $A_i \notin$ PK then $t[C_i] \geq t[C[PK]]$ (where C[PK] is the classification of the apparent primary key PK).

3.3 REFERENTIAL INTEGRITY IN MULTILEVEL RELATIONS

As discussed in the previous section, a referential integrity constraint states that a foreign key value in a referencing relation should always have a matching primary key value in the referenced relation. If the reference is between two values that are at different access classes, there is a possibility of security violation. There are certain instances of referential integrity in multilevel relations which give rise to signalling channels, i.e., cause downward flow of information. In this section we will discuss in detail the effect of enforcing referential integrity rules, and will list the requirements for having integrity and secrecy simultaneously in a multilevel relation.

According to the entity integrity constraint, if there is a multi-attribute primary key for a multilevel relation, then the primary key should be uniformly classified. The same argument is extended to the foreign keys to give the first requirement for referential integrity.

Requirement 1. The foreign key of the referencing relation must be uniformly classified (i.e., all attribute values that make up the foreign key must be assigned an identical access class).

To study the effect of the delete, insert, and update rules on referential integrity in multilevel relations, we need to identify the different possible relationships between the access class of the foreign key and the access class of the targeted primary key. The possible relationships are as follows:

1. C[FK] < C[PK]
2. C[FK] = C[PK]
3. C[FK] > C[PK]

where C[FK] denotes the access class of a foreign key value in the referencing relation and C[PK] denotes the access class of the apparent primary key in the referenced relation. These three cases have an impact on the multilevel interpretations of the single-level referential integrity rules given in the previous section. We do not need to consider the relationship where C[FK] and C[PK] are incomparable since referential integrity cannot be enforced in this case.

3.3.1 Enforcement of Integrity Rules When C[FK] < C[PK]

First consider the case when the access class of the foreign key value (i.e., the referencing tuple) is lower than the access class of the primary key value in the referenced tuple. None of the insert, delete, or update integrity rules would work for this case as there exists a signalling channel. Depending on the presence or absence of the referenced primary key value at the higher access class, a low user would be allowed or not allowed to insert a referencing tuple. This gives rise to a potential illegal flow of information from a high subject to a low subject. This channel can be better explained with the help of the following example.

Example 1 - Consider a relation schema SOD(Starship, Objective, Destination) which gives information on the name of the starship (Starship), the purpose of the flight (Objective), and the destination of the flight (Destination). Starship is the apparent primary key of the relation schema SOD, and the security classifications are assigned at the granularity of individual data elements. The hierarchical order usually followed is Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U). A user with a Secret clearance will see the entire multilevel relation SOD_S, while a user with an Unclassified clearance will see an Unclassified instance SOD_U, as shown in figure 1.

Also consider the multilevel relation schema PS (Person_Name, Starship) which contains names of crew members (Person_Name) and the starship to which the person is assigned (Starship). Person_Name is the apparent primary key of the relation schema PS, and Starship is the foreign key of PS which refers to the relation schema SOD. Typical relation instances for PS at the Unclassified level (PS_U) and Secret level (PS_S) are also shown in figure 1.

Just by looking at the two relations the Unclassified user can infer that there is a flight for the Starship "Apollo" which is classified at a higher access class. This follows from the basic referential integrity rule, as there is a Starship "Apollo" present in the referencing relation instance PS_U but not in the referenced relation instance SOD_U. This is an inference problem.

In addition, a signalling channel problem will arise if an Unclassified user attempts to insert the tuple with Starship = "Apollo" in the relation PS. The Unclassified user is allowed to insert the tuple only if there is already a tuple with Starship = "Apollo" in the relation SOD at the Secret or higher level. If there is no such tuple in relation SOD then the unclassified user's attempt is rejected, and this creates a potential signalling channel.

It has been observed that if the access class of the foreign key is lower than the access class of the referenced primary key, then there exists a channel on insertion and also there is an inference problem. This is a violation of security policy. The same problem exists for the other levels of granularity. This gives us the second requirement.

Requirement 2. The access class of the foreign key should always dominate the access class of the primary key of the referenced tuple, i.e., C[FK] ≥ C[PK].

Unclassified Instance SOD_U

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Talos U	U
Voyager U	Null U	Null U	U

Secret Instance SOD_S

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Talos U	U
Voyager U	Spying S	Mars S	S
Apollo S	Spying S	Moon S	S

Unclassified Instance PS_U

Person_Name	Starship	TC
James Kirk U	Enterprise U	U
John Spock U	Null U	U
Tim Morris U	Apollo U	U

Secret Instance PS_S

Person_Name	Starship	TC
James Kirk U	Enterprise U	U
John Spock U	Voyager S	S
Tim Morris U	Apollo U	U

Figure 1. Multilevel Relations SOD and PS

3.3.2 Enforcement of Integrity Rules When C[FK] = C[PK]

When the access class of the referencing tuple and the access class of the referenced tuple are the same, all the integrity rules are usable without modification because this case can be considered to be equivalent to relationships occurring in a single-level database. All the insert, delete, and update rules have been formulated for single-level relationships and do not cause violations of either the security rules or the integrity rules for multilevel data bases. This fact has been acknowledged in the SeaView model [LUNT90] and the referential secrecy model given by [BURN88], although Burns also suggests selective enforcement of referential integrity in those cases where the access class of both the referencing relation and the referenced relation are not the same if the signalling channel either can be monitored or is deemed not to cause a serious threat.

So it can be concluded that if the access class of the foreign key value is same as the access class of the referenced primary key value then all integrity rules for update, delete, or insert can be used without violation of security constraints for any level of security labeling.

3.3.3 Enforcement of Integrity Rules When C[FK] > C[PK]

The last case, where the access class of the referencing tuple dominates the access class of the referenced tuple, needs to be considered now. Each of the integrity rules for insert, delete, and update will be considered individually, and the effect of strict dominance of the referencing tuple's access class over that of the referenced tuple will be investigated for non-polyinstantiated multilevel relations and polyinstantiated multilevel relations. It is necessary to include the case of polyinstantiated relations here because it has been observed that polyinstantiation causes semantic ambiguity in a DBMS providing referential integrity control, when C[FK] > C[PK].

For both polyinstantiated and non-polyinstantiated relations, the validity of the referential integrity rules will be studied for multilevel relations with element-level granularity only. The behavior of the integrity rules at other

levels of security labeling are discussed in detail in [DOSH91]. For a quick and short reference we give the results of the study for all levels of granularity in the conclusion of this paper.

3.3.3.1 Polyinstantiated Multilevel Relations

First we discuss the referential integrity rules when enforced in polyinstantiated multilevel relations with element-level granularity. For this discussion, example instances of the relations SOD and PS incorporating a polyinstantiated tuple have been adopted. The relations SOD and PS with element level granularity, look as shown in figure 2. Tuple numbers have been added for easy references in the following example.

To understand the problems in providing referential integrity control in polyinstantiated multilevel relations with element-level granularity, each of the integrity rules will be considered individually using the relations SOD and PS given in figure 2. It should be noted that the primary key for both the relations is the attribute alone without its classification level. It is observed in the discussion that for all the integrity rules, i.e., the insert rule, the delete rules, and the update rules, integrity is best maintained when the access class is included in the primary key which is equivalent to having the access class of the foreign key equal to the access class of the targeted primary key.

SOD

No.	Starship	Objective	Destination	TC
1	Enterprise U	Exploration U	Talos U	U
2	Enterprise S	Spying S	Mars S	S

PS

No.	Person_Name	Starship	TC
3	James Kirk U	Enterprise U	U
4	John Spock S	Enterprise S	S

Figure 2. Relations SOD and PS with Polyinstantiation

1. *Insert Rule* - The insertion of tuples in multilevel relations with element-level labeling while maintaining referential integrity, depends on the order of insertion. There may arise situations where there are signalling channels. For instance, suppose that first tuples 1 and 2 are inserted in the relation SOD and then tuples 3 and 4 are inserted in relation PS. Tuple 3 in relation PS should not be inserted as there exists a referenced tuple 2, with a primary key value having an access class higher than the access class of the foreign key value in tuple 3. But as there also exists a matching tuple 1 in relation SOD, tuple 3's insertion should not be rejected. Therefore there is a conflict. This can be resolved by having the access class of the primary key value included in the apparent primary key. In that case the access class will also be included in the foreign key, and the insertion of a referencing tuple will be accepted only if the access classes also match.

2. *Delete Rules* - Assuming that the insertion of the rows succeeds, it is observed that there is referential ambiguity when an attempt is made to delete the primary key value [GAJN88]. Suppose that all four rows in relations SOD and PS have been successfully inserted as shown in figure 2. An Unclassified user makes an attempt to delete the tuple 1 from the relation SOD. For either CASCADE or NULLIFIES-delete, though, the deletion is permitted it is not clear which referencing tuple to delete or set null in the relation PS. Tuples 3 and 4 in the relation PS both reference Starship = 'Enterprise.' If the primary key of relation SOD included its access class, then it would have been obvious that it is tuple 3 which has to be deleted or have its foreign key set to null, when the Unclassified user deletes tuple 1.

3. *Update Rules* - The update rules work in a similar fashion to the delete rules. As in the case of delete rules, there is a referential ambiguity when the primary key only includes the apparent PK. The confusion can be eliminated by including the access class in the apparent primary key.

From the discussion above, it can be concluded that in polyinstantiated relations with element-level granularity, the access class of the apparent primary key must be included in the primary key. Having the access class included in

the primary key is the same as having the access class of the foreign key equal to the access class of the primary key, i.e., C[FK] = C[PK].

Requirement 3. For polyinstantiated multilevel relations with element-level granularity, the access class of the foreign key must be same as the access class of the referenced primary key.

3.3.3.2 Non-Polyinstantiated Multilevel Relations

In this section, the discussion is on whether or not referential integrity can be provided in non-polyinstantiated relations, when the access class of the foreign key is higher than the access class of the apparent primary key. Without polyinstantiation, referential ambiguities cannot exist; however, conflicts may arise that prevent certain referential integrity rules from operating successfully when the access classes of the foreign key and targeted primary key differ.

Each of the integrity rules for insert, delete, and update defined in section 2.2 is examined individually and conditions violating the security constraints are investigated.

1a. *RESTRICTED-delete Rule* - The RESTRICTED-delete rule states that the referenced primary key tuple cannot be deleted as long as there is a corresponding referencing tuple somewhere in the database. For instance, consider the relations SOD and PS given in figure 3.

SOD

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Talos U	U
Voyager U	Spying S	Mars S	S
Apollo S	Spying S	Moon S	S

PS

Person_Name	Starship	TC
James Kirk U	Enterprise U	U
John Spock S	Voyager S	S

Figure 3. Relations SOD and PS without Polyinstantiation

In relation PS, the access class of the foreign key value Starship = "Voyager" dominates the access class of the primary key value in the relation SOD. According to the RESTRICTED delete rule, as long as there is a tuple having foreign key value "Voyager" in PS, the tuple with primary key value "Voyager" cannot be deleted from the relation SOD. This gives rise to a signalling channel, as there is a downward flow of information from the high level referencing foreign key to the low level subject attempting to delete the tuple containing the primary key. Therefore, the RESTRICTED-delete Rule violates secrecy when the access class of the foreign key dominates the access class of the primary key of the referenced tuple.

1b. *CASCADES-delete Rule* - As discussed earlier, the CASCADES-delete rule states that when a primary key value is deleted, all corresponding referencing tuples should also be deleted. For instance, if the tuple in SOD with Starship = "Voyager" is deleted at the Unclassified level, then the tuple in PS with Starship = "Voyager" at the Secret level will also be deleted. This action causes a signalling channel. In the relation instance of PS given in figure 3, Person_Name = "William Spock" is at Unclassified level in the tuple containing Starship ="Voyager" at Secret level. When the tuple is deleted as a result of the CASCADES-delete action, an Unclassified user will come to know that there existed a starship "Voyager" at some higher level. Hence, the CASCADES-delete rule fails for the case where C[FK] > C[PK]. For all other levels of granularity, CASCADES-delete rule remains valid [DOSH91].

1c. *NULLIFIES-delete Rule* - The NULLIFIES-delete rule states that when a primary key value in the referenced relation is deleted, then the corresponding foreign key values in the referencing relation should be set to null. As an example the value of Starship = "Voyager" in relation PS is set to null when the tuple in SOD for Starship =

"Voyager" is deleted. This does not conflict with the security constraints, as long as *write-ups* are allowed, since there would not be a downward flow of information.

From the discussion above, unlike polyinstantiated relations with element-level labeling there is no referential ambiguity while deletion and the following requirement can be inferred for the case when the access class of the foreign key value dominates the access class of the primary key value:

> **Requirement 4.** In relations with element-level granularity, when the access class of the foreign key value is higher than the access class of the primary key value, only NULLIFIES_delete rule and SET DEFAULT-delete rule can be used as delete options to specify referential constraints.

2a. *RESTRICTED-update Rule* - Consider the relations SOD and PS given in figure 3. In relation PS, the access class of the foreign key value Starship = "Voyager" dominates that of the primary key value in the relation SOD. According to the RESTRICTED-update rule as long as there is the tuple having foreign key value "Voyager" in PS, the primary key value "Voyager" cannot be updated in the relation SOD. This gives rise to a signalling channel as there is flow of information from the high level referencing foreign key to the low level subject attempting to delete the tuple containing the the primary key. Therefore, the RESTRICTED-update Rule will not work if the access class of the referencing tuple dominates the access class of the referenced tuple.

2b. *CASCADES-update Rule* - The CASCADES-update Rule states that if the primary key value Starship = "Voyager" in SOD is updated, then the foreign key value Starship = "Voyager" in PS will also be updated. This action will not cause a signalling channel and will also maintain integrity, irrespective of the fact that the access class of the foreign key value dominates the access class of the primary key value.

2c. *NULLIFIES-update Rule* - The NULLIFIES-update Rule specifies that the value of Starship = "Voyager" in relation PS be set to null if Starship = "Voyager" in SOD is updated. This does not conflict with the security constraints, as long as *write-ups* are allowed, since there would not be a downward flow of information.

From the discussion for each of the update rules the requirement for the specification of the update rules is the following:

> **Requirement 5.** When the access class of the foreign key value is higher than the access class of the referenced primary key value in relations with element-level granularity, then only CASCADES-update, NULLIFIES-update, or SET DEFAULT-update should be used. The RESTRICTED-update rule violates the secrecy constraints.

3. *INSERT Rule* - The insert rule for integrity states that the insertion of a foreign key value should comply with the nulls rule and the basic referential integrity rules; that is, each foreign key value in the referencing relation should have an identical primary key value in the referenced relation. In multilevel databases, it is important to ensure that there is no downward flow of information when the insertion is made. If the access class of the referencing tuple (foreign key value) dominates the access class of the referenced tuple (primary key value) then there is no such possibility of a signalling channel. If a higher user attempts to insert a foreign key value, the user's insertion is accepted or rejected based on the presence or absence of the referenced value at the lower access class. There will be only an upward flow of information, which means that both integrity and security rules are satisfied. It should be noted that, unlike with polyinstantiated relations, there is no confusion while inserting the rows in the two relations.

> **Requirement 6.** For the insert rule to be valid without violation of security or integrity constraints, the access class of the foreign key value should dominate the access class of the referenced primary key value in relations with element-level granularity.

4. CONCLUSIONS

Entity integrity and referential integrity are two important integrity constraints that should be enforced by the DBMS. Referential integrity is the most important inter-relation integrity constraint in the relational data model. In

this paper, the integrity constraints have been discussed for single-level relations, then the concepts have been extended to the multilevel world.

The extension of the concepts of referential integrity from single-level relations to multilevel relations is not straightforward. This is because restrictions are needed to provide referential integrity control in MLS DBMSs without compromising secrecy. The basic requirement for referential integrity is that each referencing foreign key value must have an identical target primary key value in the referenced relation. An additional requirement for multilevel relations is that the foreign key and the primary key should be uniformly classified, i.e., all attributes included in the key should have the same access class.

From the discussion in the paper, it can be concluded that enforcing referential integrity when the access class of the foreign key is equal to the access class of the referenced primary key is simple and without any ambiguity. All integrity rules apply in this case, whether the relations allow polyinstantiation or not. In fact, when polyinstantiation is allowed, the access class of the primary and foreign key values must be included as part of the key to disambiguate references and allow the referential integrity rules to be enforced.

Referential integrity completely fails when the access class of the foreign key does not dominate the access class of the primary key. When the access class of the foreign key strictly dominates the access class of the referenced primary key, some of the integrity rules for actions to be taken on deletion or modification of a key value apply, with some differences based on labeling granularity. A summary of the validity of different referential integrity rules for different levels of granularity is given in table 1.

Table 1. Summary of Validity of Integrity Rules for C[FK] > C[PK]

Level of Granularity	**Element**		**Tuple**		**Attribute**	**Relation**
Polyinstantiated	**Poly.**	**Not Poly.**	**Poly.**	**Not Poly.**	**Not Poly.**	**Not Poly.**
Insert Rule	Invalid	OK	Invalid	OK	OK	OK
Delete Rules:						
Restricted-delete	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
Cascades-delete	Invalid	Invalid	Invalid	OK	OK	OK
Nullifies-delete	Invalid	OK	Invalid	OK	OK	OK
Set-default-delete	Invalid	OK	Invalid	OK	OK	OK
Update Rules:						
Restricted-update	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
Cascades-update	Invalid	OK	Invalid	OK	OK	OK
Nullifies-update	Invalid	OK	Invalid	OK	OK	OK
Set-default-update	Invalid	OK	Invalid	OK	OK	OK

REFERENCES

[BURNS88] Rae K. Burns, "Referential Secrecy," *Proceedings of the IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, 1990.

[DATE90] C.J. Date, "Referential Integrity and Foreign Keys: Further Considerations," in *Relational Database Writings 1985-1989*, Addison-Wesley, 1990, pp. 99-184.

[DENN87] Dorothy E. Denning and Teresa F. Lunt, Roger R. Schell, Mark Heckman, and William R. Shockley, "A Multilevel Relational Data Model," *Proceedings of the 1987 IEEE Symposium on Security and Privacy* , IEEE Computer Society Press, 1987, pp. 220-234.

[DOSH91] Vinti M. Doshi and Sushil Jajodia, "Referential integrity in multilevel secure database management system," The MITRE Corporation, 1991.

[GAJN88] George E. Gajnak, "Some Results from the Entity/Relationship Multilevel Secure DBMS Project," *Proceedings of the Fourth Aerospace Computer Applications Conference*, IEEE Computer Security Press, 1988, pp. 66-71.

[JAJO90] Sushil Jajodia and Ravi Sandhu, "Polyinstantiation Integrity in Multilevel Relations," *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, California, May 1990, pp. 104-115.

[JAJO91a] Sushil Jajodia and Ravi Sandhu, "Toward a Multilevel Secure Relational Data Model," *Proceedings of ACM SIGMOD International Conf. on Management of Data*, Denver, Colorado, May 29-31, 1991, pp. 50-59.

[JAJO91b] Sushil Jajodia and Ravi Sandhu, "Enforcing Primary Key Requirements in Multilevel Relations," Proceedings of the Fourth RADC Workshop on Multilevel Database Security, Little Compton, Rhode Island, April 1991.

[LUNT90] Teresa F. Lunt, Dorothy E. Denning, Roger R. Shell, Mark Heckman, and William R. Shockley, "The SeaView Security Model," *IEEE Transactions on Software Engineering*, Vol. 16, No. 6, June 1990, pp.593-607.

EVOLVING CRITERIA FOR EVALUATION: THE CHALLENGE FOR THE INTERNATIONAL INTEGRATOR OF THE 90s.

Virgil Gibson
839 Elkridge Landing Road
Suite 106, Box 24
Linthicum, MD 21090

Joan Fowler
2411 Dulles Corner Park
Suite 500
Herndon, VA 22071

Abstract

This paper presents a comparison of three international security criteria and contrasts their approaches. It demonstrates the integrator's perspective of the implications for designing solutions to meet worldwide information protection needs. The implied and stated use of each of the criteria is included in this paper with a description of the barriers to understanding between the criteria. A high level comparison of the approaches taken in the three criteria is presented. From the system integrator's perspective, the causes for the ambiguity between these criteria are discussed, with suggestions for the international community.

Introduction

The United States Department of Defense (DoD) published in 1985 the DoD Trusted Computer System Evaluation Criteria (TCSEC) [3] which is the seminal work in detailing the criteria which guide buyers and sellers in the computer security arena. The TCSEC, and its interpretations; the Trusted Network Interpretations (TNI) [7], the Trusted Database Interpretation (TDI) [8], and the Computer Security Subsystem Interpretation (CSSI) [9], form the nucleus around which the U.S. Trusted Product Evaluation Program has developed.

Recently, two new criteria have appeared on the world scene which are to be used in the evaluation of security applications of computer technology. In December 1990, the Canadian System Security Centre (CSSC) released the Canadian Trusted Computer Product Evaluation Criteria (CTCPEC), Version 2.0. The CTCPEC was revised in April, 1992, as Version 3.0. [2] In June 1991, the Harmonized Criteria of France, Germany, The Netherlands and the United Kingdom (UK) was published as the Information Technology Security Evaluation Criteria (ITSEC), Version 1.2 [5].

This paper is divided into two overall segments. The first segment is a description of the three criteria: their intended audiences, life cycle use, and approaches. The second segment describes how the international systems integrator is involved in the situation. This second segment also provides the system integrator's perspective of the first segment: the ambiguities caused by the three international criteria, and suggestions for the future.

Stated or Implied Use of Each Criteria

Before discussing the intended use of each of the criteria, the definition of a couple of terms is in order. For the purposes of this paper, a product "is a hardware and/or software package that can be bought off the shelf and incorporated into a variety of systems." A system "is a specific [data processing] installation with particular purpose and known operational environment" [5].

Intended Audience for the Criteria

All three of the criteria have a common understanding of these two terms, product and system. However, the three criteria are not intended to be used in the same context with regard to the two terms.

The target for evaluations using the U.S. TCSEC has historically been primarily products. Its focus is on industry, in that it is an incentive to vendors to develop trusted products. An entire series of guidelines have been developed to aid industry in the use of the criteria. The TCSEC is intended to provide guidance to the commercial world on the development of trusted products, and provide a direction for the growth of each individual product. Industry may choose to grow their product toward more security features or more assurance in these features.

The Canadian CTCPEC specifically states that the target for its evaluation is products. However, the focus of the CTCPEC is on government. The criteria does not provide clear direction for the growth of commercial products. It provides only the criteria by which a product may be evaluated once it is developed. However, the CTCPEC very conveniently allows a vendor to develop a product, and then add to the assurances of the product by improving the documentation in a growth fashion.

The European ITSEC specifically maintains that its target for evaluation is both products and systems. The ITSEC states that "it is important for the sake of consistency that the same security criteria are used for both products and systems; it will then be both easier and cheaper to evaluate systems containing products which have already been successfully evaluated." [5] Because of the loose nature of the ITSEC approach, it also does not present a clear growth direction for the security features of products. However, like the CTCPEC, vendors may build a product/system and then add to the assurances of the product/system by improving the documentation and receiving a higher rating for assurance.

Barriers to Understanding

There are further barriers to understanding of the three criteria. The use of terminology in the three criteria is one of these barriers. Certain actions in the overall evaluation process are not universally identified in the three schemes. To illustrate this point, a simile of the process by which a household appliance receives the Underwriters Laboratories (UL) Seal of Approval will be used. The appliance is developed by a vendor and submitted to UL for testing. UL initially may test the appliance in a laboratory. This is the same as the evaluators scrutinizing a product "from a perspective that excludes consideration of a specific application environment." [2] In the U.S. and Canada, this process is referred to as the Evaluation performed by the National Computer Security Center (NCSC) and the CSSC respectively. However, since the ITSEC is aimed at both products and systems, there is not a clear distinction between this first type of evaluation and those performed for specific application environments.

Returning to the simile, once the UL has tested an appliance in the laboratory, it may be tested for application to some specific representative types of households. This is similar to the "assessment to determine whether appropriate security measures have been taken to permit the system to be used operationally in a specific environment." [3] This second type of evaluation is called the Certification evaluation under the TCSEC. In the CTCPEC, the exact same type of evaluation is termed a Risk Assessment. Again, since the ITSEC does not differentiate between its use for systems and products, it only recognizes the single type of evaluation.

The final step in the process of the appliance testing is the award of the UL Seal of Approval. This is awarded after the evaluation; and an official authorization is given that the appliance meets the standards for the Seal of Approval. In the TCSEC, the "official authorization that is granted to an [Automatic Data Processing] ADP system to process sensitive information in its operational environment" [3] is the formal approval/accreditation procedure, usually referred to as an accreditation. The CTCPEC refers to the same procedure as the formal approval. The ITSEC indicates that national certification bodies "will award certificates to confirm the rating of the security." [5] In the ITSEC, the process to award these certificates is referred to as the certification process.

This simile of household appliance UL testing demonstrates that terminology is a confusion factor for the systems integrator using all or a combination of the three criteria. Since evaluation, risk assessment, certification, final approval, and accreditation can mean different processes within the approval of systems and products, the integrator must be aware of the criteria and audience when using each of these terms.

Life Cycle Use of the Criteria

At the current time, the practical use of the TCSEC is much more specified than that of the other two criteria. Since the TCSEC has been available longer, this situation is not surprising. The NCSC has developed guidelines for vendors on the use of the TCSEC for product evaluation and evaluation maintenance: the Trusted Product Evaluations, A Guide for Vendors [10]; and the Rating Maintenance Phase, Program Document [11]. The first describes the procedure which the vendor should use to take a product through evaluation, and the second defines the procedure to reevaluate subsequent releases of evaluated products.

Neither the CTCPEC nor the TCSEC discuss the practical use of the criteria. The ITSEC acknowledges that there needs to be national certification bodies to perform evaluations and a national procedure for the maintenance of certified ratings following changes to an evaluated target. However, the details of both of these procedures are indicated as being beyond the scope of the ITSEC. This spring the Information Technology Security Evaluation Manual (ITSEM) was released for comments in draft version 0.2. [6] This document is meant to "harmonize existing security evaluation methods in each of the four countries in order to ensure that national evaluation methods conform to a single philosophy". This recent development is a major step toward standardization of the use of the ITSEC. The latest version of the CTCPEC indicates that it was written in such a way to preclude the need for interpretations. However, only practice will determine the procedures for the use of that criteria.

Approach Comparison

The schemes defined in each of the criteria are different in some ways, and the same in others. However, the underlying principle is common to the three criteria. This underlying principle is that there are security features (e.g., access control, auditing) which are required to be available in products/systems. Additionally, there are assurances (e.g., documentation, testing) which must be used to prove that the features are performing completely, correctly, and consistently. All three of the criteria agree on these concepts. However, the packaging of these concepts and some of the details of implementation of these principles are drastically different. The similarities and differences are addressed here to highlight the differences which will cause a barrier to international acceptance of products/systems and illustrate the knowledge which an international integrator needs to compete.

The U.S. TCSEC Approach

The TCSEC has specified the set of functional features and a set of the type of associated assurances which a product must possess for each class level. These features and assurances are bundled together into a single class. There are four possible divisions containing seven classes in the TCSEC scheme. These divisions are Minimal Protection (D), Discretionary Protection (C1 and C2), Mandatory Protection (B1, B2, and B3), and Verified Protection (A1). A target (product) must meet all of the requirements of a class to be assigned the class level. If one of the functions or assurances of a particular class is not available for the target, the next lower class level with which the target complies is assigned to the target.

The Canadian CTCPEC Approach

The CTCPEC scheme is separated into criteria of security functionality features and assurances. Unlike the TCSEC, the functional features are not grouped together with the assurance requirements. There are five criteria: Confidentiality, Integrity, Availability, Accountability, and Assurance. Each criteria is further divided into divisions and levels. The Confidentiality criteria is decomposed into: covert channels (CC-0, CC-1, CC-2, and CC-3); discretionary confidentiality (CD-0, CD-1, CD-2, CD-3, and CD-4); mandatory confidentiality (CM-0, CM-1, CM-2, CM-3, and CM-4); and object reuse (CR-0 and CR-1). The Integrity criteria is divided into: discretionary integrity (ID-0, ID-1, ID-2, ID-3, and ID-4); mandatory integrity (IM-0, IM-1, IM-2, IM-3, and IM-4); physical integrity (IP-0, IP-1, IP-2, IP-3, and IP-4); rollback (IR-0, IR-1, and IR-2); separation of duties (IS-0, IS-1, IS-2, and IS-3) and self testing (IT-0, IT-1, IT-2, and IT-3). The Availability criteria is divided into: containment (AC-0, AC-1, AC-2, and AC-3); robustness (AR-0, AR-1, AR-2, and AR-3); and recovery (AY-0, AY-1, AY-2, and AY-3). The Accountability criteria is partitioned into: identification and authentication (WI-0, WI-1, and WI-2); audit (WA-0, WA-1, WA-2, WA-3, and WA-4); and trusted path (WT-0, WT-1, and WT-2). Finally, the

Assurance criteria consists of trust (T-0, T-1, T-2, T-3, T-4, T-5, T-6, and T-7). Tables 1 through 4 illustrates the CTCPEC profiles which correspond to the TCSEC Classes C2 through B3 respectively. There is no correlation to these profiles and the associated levels of trust within the corresponding TCSEC Classes. An equivalent TCSEC profile does not imply that a TCSEC rating meets the profile. Hence, these are to be considered as one way mappings.

As in the TCSEC, a target (product) must meet all of the requirements for a specific level, otherwise it is assigned the lowest class with which the target complies completely. Each criteria division contains a level designated "0". This level is reserved to targets which have been evaluated but failed to meet the requirements of any of the higher levels for the category division.

Table 1. CTCPEC Profile Equivalent to TCSEC Class C2

Functionality	Division/Mechanism	Level
Confidentiality	Discretionary	2
	Object Reuse	1
Integrity	Discretionary	1
	Separation of Duties	1
	Self Testing	1
Accountability	I & A	1
	Audit Level	1

The European ITSEC Approach

The ITSEC also has separated the functionality into a separate rating from the assurances or, as stated in the ITSEC, the effectiveness and correctness aspects of assurance. There are ten example functionality classes. The first five are closely tied to the TCSEC classes (F-C1, F-C2, F-B1, F-B2, and F-B3). Table 5 maps the ITSEC classes to the TCSEC classes. This mapping is a general guide, the two criteria schemes do not directly correspond to each other.

The other five classes are new in the ITSEC: high integrity requirements (F-IN); high requirements for availability of complete or special functions of the target (F-AV); high requirements for data communication integrity (F-DI); high demands for data communication confidentiality (F-DC); and networks with high demands on confidentiality and integrity of information (F-DX). These classes are examples and not obligatory. They can only be used if the target (product or system) contains all aspects of the class. A target may reference one or more of these example functionality classes to define part or all of its functions. As an alternative to the use of the example pre-defined functionality classes, the sponsor of evaluation can specify the security enforcing functions of the target.

Table 2. CTCPEC Profile Equivalent to TCSEC Class B1

Functionality	Division/Mechanism	Level
Confidentiality	Discretionary	2
	Mandatory	2
	Object Reuse	1
Integrity	Discretionary or Mandatory	1
	Separation of Duties	1
	Self Testing	1
Accountability	I & A	1
	Audit Level	1

There are seven possible correctness levels (E0, E1, E2, E3, E4, E5, and E6) described in the ITSEC. In addition, following the evaluation of the correctness, an assessment of effectiveness based on a vulnerability analysis of the target is performed. There is a pass/fail designation of the evaluation on effectiveness grounds.

The ITSEC approach is more flexible, and more open to interpretation by all of the national certification bodies which will perform the evaluations. This flexibility may limit the ability for any future reciprocal recognition of certifications. The actual practice in the use of the criteria in the future will determine the feasibility of this

approach. For the systems integrator, this approach has the potential to evolve into four different practical usages, one for each of the involved countries: United Kingdom, The Netherlands, France and Germany.

Table 3. CTCPEC Profile Equivalent to TCSEC Class B2

Functionality	Division/Mechanism	Level
Confidentiality	Covert Channels	1
	Discretionary	2
	Mandatory	3
	Object Reuse	1
Integrity	Discretionary or Mandatory	1
	Separation of Duties	2
	Self Testing	1
Accountability	I & A	1
	Audit Level	1
	Trusted Path	1

Use of the Three Criteria

For an example of the rating of a product under each of the schemes, we selected a fictional product that is an M-Component (Mandatory Access Control) under the TNI with a rating of TCSEC Class B1 M-Component. The TNI "allows for the evaluation of components which in and of themselves do not support all the policies required by the TCSEC" but which can be reused "in different networks without the need for a re-evaluation of the component." [7]

This same product when rated under the CTCPEC would have a rating of CM-1, CR-1, WI-3, and T-3. This designation is a clear correspondence between the two criteria, TCSEC (TNI) and the CTCPEC. The correspondence is not as clear to the ITSEC scheme. The closest ITSEC example rating is F-B1 and E3. However, a F-B1 has requirements which do not correlate to the M-Component designation of the TNI (e.g., Identification and Authentication, "A.20 The TOE shall uniquely identify and authenticate users" [5]). Further, there are requirements for F-B1 which are not designated in the U.S. scheme for a B1 M-Component, reference [1] page 48.

Approach Conclusion

This discussion of the differing terminologies, requirements, and approaches of the three criteria must lead to the conclusion that there is no consistency between them. This inconsistency leads to confusion in the systems integration community among others. The potential effects that this confusion will have on this community are discussed below.

Systems Integrator Perspective

Systems vendors and integrators, who expect to survive through the decade of the 90s, will have to contend with the trusted systems criteria discussed above. Their differing requirements and approaches, and the implications of their use will determine international competitiveness.

What is a Systems Integrator?

In this paper, the term System Integrator is defined as follows: A systems integrator provides the expertise to cost effectively bring together divergent products from multiple product lines to solve a specific operational problem in a specific installation. In the case where the problem includes protection of information, some of the products will have security functionality, and will likely have been evaluated by one of these international schemes.

Systems integrators, as defined for this paper, typically respond to Requests for Proposals (RFPs) and Invitation for Bids (IFBs) from Governments and related Organizations, such as the North Atlantic Treaty Organization (NATO), solving a specific problem (part of which is assumed to include security) in a given

Table 4. CTCPEC Profile Equivalent to TCSEC Class B3

Functionality	Division/Mechanism	Level
Confidentiality	Covert Channels	1
	Discretionary	3
	Mandatory	3
	Object Reuse	1
Integrity	Discretionary or Mandatory	1
	Separation of Duties	2
	Self Testing	1
Availability	Recovery	1
Accountability	I & A	1
	Audit Level	2
	Trusted Path	2

environment. Procurement documents must be developed such that vendors with solutions based on approved products from any evaluation scheme could compete. Such procurement documents would contain phrases such as: the proposed solution shall comply with all requirements for ITSEC F-B1, E3. Systems integrator teams would then design a solution which was composed of subsystems and products which met the requirement in the most cost effective combination.

Of course, the solution to any large requirement usually requires combining (evaluated) security products into more complex systems. The analysis of the total problem, with security considerations, requires that the security team be conversant with the entire architecture, each of the evaluation schemes, and the products/technology evaluated under each scheme.

When the requirement is stated in terms like: the offerors' solution must be capable of evaluation at the TCSEC C2 level, the integrator's problem is compounded. Products which have been evaluated according to one scheme may not be acceptable, or meet the analogous level in the others. The team must consider the most cost-effective path to satisfying the requirement and that path may require including an evaluation (or re-evaluation) of a product or assurance documents in the overall cost.

What Do Multiple Evaluations Mean to Systems Integrators?

Vendors having products which have been evaluated on more than one scheme will, of course, be reluctant to draw distinctions among criteria. They are in business to sell products, not necessarily to solve a given specific operational problem.

Micronyx, an international vendor of a product, TriSpan, have listings in the UK Certified Security Products List (UKSP), and the U.S. Evaluated Products List (EPL). The product was advertised in Infomatics [4] as UK Government Certified and U.S. NCSC Certified. Indeed, the 1 October 1991 UKSP lists TriSpan version 1.2130 as meeting UKL2 (Independently Tested). The U.S. NCSC EPL-SUM-89/007, however, gave the product an overall rating of TCSEC Class D, for Identification and Authentication (I&A), Discretionary Access Control (DAC), and Audit, stating that it does not meet the assurance and documentation requirements for a higher rating.

A trusted systems integrator contemplating using such a product to meet an overall requirement of any of the equivalent evaluation ratings, as described

Table 5. ITSEC to TCSEC Mappings

TCSEC Class	ITSEC Class
D	E0
C1	F-C1, E1
C2	F-C2, E2
B1	F-B1, E3
B2	F-B2, E4
B3	F-B3, E5
A1	F-B3, E6

earlier, must be able to accurately estimate the costs to develop assurance and documentation which may be required by the accrediting or approving official.

Causes for Ambiguity

An international systems integrator is faced with ambiguity caused by several factors. Each of the principals involved with these criteria has autonomy to construct evaluation schemes, and maintain separate lists of evaluated products. Currently Canada, the UK and the U.S. have such schemes in place and soon there likely will be six lists.

Technology Export Restrictions

Six lists, mostly kept by the Intelligence community, implies loads of red tape to export technology, even if the techniques are already in documented use in the intended market. A striking example of this now is the Data Encryption Standard (DES) controversy. All the major U.S. vendors of systems like International Business Machine Corporation (IBM), Digital Equipment Corporation (DEC), Hewlett Packard (HP), etc, incorporate DES in the security functions of their offerings. The DES modules must be removed, or very costly negotiations must be engaged to allow selling their products outside the U.S., even though DES is well documented, and other sources for it exist outside the U.S. Similar restrictions are being applied to any products rated B3 or higher under the TCSEC.

Language Translation

Additionally, the six lists are, of course, maintained in the native language of the keepers. The nuances described earlier in this paper are exacerbated with translation. Each scheme is likely to have an array of "interpretations and guidelines" such as now exist in the U.S. scheme. Both the CTCPEC and ITSEC contain requirements which are not found in the TCSEC, and will therefore, require additional explanation of how these requirements could be met by U.S. EPL products.

Standards and Security Requirements

The inherent ambiguity which must be resolved by system integrators is aggravated by divergent standards. Security products are built to comply with criteria, not International Standards Organization (ISO) standards. When both standards and evaluation ratings are stated as requirements in procurements, the system integrator must usually make a compromise. A detailed technical knowledge of both will be required to develop a cost effective solution which complies with the intent of standards and security requirements.

Generational Problems

The multiple evaluation syndrome is compounded by what may be termed the "generational" problem. As criteria and schemes evolve, and products are evaluated, systems integrators must know when the evaluation was completed in order to assess its usefulness. The UK evaluation scheme predates the ITSEC. The CTCPEC states that revisions may be annual. (In fact, a major revision to the CTCPEC was released between the time that this paper was conceptualized and written, requiring major changes to this paper which superficially describes the approach. A wonderful example of the pitfalls and frustrations caused by generations of criteria.) Additionally, the U.S. is developing a new federated criteria. It is clear that procurement professionals won't stay abreast of these changes. Several recent U.S. procurements required solutions to comply with the 1983 version of TCSEC, and it is very difficult to ascertain which products were evaluated using that criteria. Product vendors will naturally strive to get commitments from evaluators to freeze requirements before evaluations are begun, to avoid moving targets. Subsequent users of those products will necessarily need to know this information when developing specifications for systems.

Suggestions for Community

This paper has surfaced a number of issues which result in the following suggestions for the community. The community includes product vendors, evaluators, national certification bodies, procurement professionals, accreditation officials and systems integrators -- to name a few.

"Keeper" of Evaluated Products Lists

Find a "keeper" for the lists of nationally evaluated products. Everyone can profit from others' work if some registry existed which points to evaluations completed internationally. Some international organization, such as the United Nations, could be approached to maintain this registry. (It appears that in the new world order, NATO may be looking for some jobs to do!)

Reciprocal Evaluations

Develop reciprocal evaluations using the three criteria. If a registrar could be found, this may open the way for the negotiation of international mutual recognition of evaluations. There is some work going on now in this regard between the TCSEC and the CTCPEC, or U.S. and Canada. In the near term, a single list of evaluated products is beyond our reach. But imagine the benefits to be gained by having a coordinated worldwide list of evaluated security products!

International Standards for Trusted Systems Criteria

Develop an international standard for trusted systems criteria. Even better than a single list of products evaluated against several criteria would be a single international trusted systems criteria standard. Is anybody trying to coordinate the various trusted systems criteria and the ISO? Clearly the community will profit if eventual ISO standards in security services can be provided by evaluated products.

Universally accepted standards are vendor driven. Vendors naturally push the standards which they do offer or can offer in their own products. But, having several international security criteria, the vendors are not going to be economically able to support all of the criteria. It is therefore important to have this criteria coordinated closely with the international standards committees.

Evolution of Technology

Perform more analysis, such as the paper in 14th National Computer Security Conference [1]. This paper compares the requirements of TCSEC Class B3 and ITSEC example class F-B3, E5. The comparison of specific products to potential evaluations (Targets of Evaluation) sheds a much needed light on differences and similarities in these criteria. The real burden is on trusted systems professionals to stay abreast of this evolution in technology, and attempt to inform the rest of the community. Such participation in the process results in more consistent practices internationally and diminished ambiguity in reporting results.

Bibliography

[1] Branstad, Dr. Martha, et al, "Apparent Differences between the U.S. TCSEC and the European ITSEC", Proceedings: 14th National Computer Security Conference, Vol 1, page 45-58, October 1991.

[2] Canadian System Security Centre, Communications Security Establishment, Government of Canada, The Canadian Trusted Computer Product Evaluation Criteria (CTCPEC), Version 3.0., April 1992.

[3] Department of Defense, Trusted Computer System Evaluation Criteria (TCSEC), DoD 5200.28-STD, December 1985.

[4] Infomatics, "Security Survey", VNU Business Publications, Vol. II, No. 7, page 74, July 1990.

[5] Information Technology Security Evaluation Criteria (ITSEC), Provisional Harmonized Criteria, Version 1.2. 28 June 1991.

[6] Information Technology Security Evaluation Manual (ITSEM), Commission of the European Communities, Draft V0.2, 1992.

[7] National Computer Security Center, Trusted Network Interpretation of Trusted Computer System Evaluation Criteria, NCSC-TG-005, 31 July 1987.

[8] National Computer Security Center, Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria, NCSC-TG-021, Version 1, April 1991.

[9] National Computer Security Center, Computer Security Subsystem Interpretation of Trusted Computer System Evaluation Criteria, NCSC-TG-009, Version 1, 16 September 1988.

[10] National Computer Security Center, Trusted Product Evaluations, A Guide for Vendors, NCSC-TG-002, Version 1, 22 June 1990.

[11] National Computer Security Center, Rating Maintenance Phase, Program Document, NCSC-TG-013, Version 1, 23 June 1989.

An Example Complex Application for High-Assurance Systems

Frank L. Mayer*

The Aerospace Corporation
955 L'Enfant Plaza, SW
Washington, DC 20024
(mayer@aero.org)

Steven J. Padilla

SPARTA, Inc.
9861 Broken Land Parkway
Columbia, MD 21046
(smaug@columbia.sparta.com)

Abstract

A challenging problem is the development of sophisticated applications for multilevel secure systems without the undesirable introduction of application code in the trusted portion of the system. This paper discusses issues related to such applications and presents a straw man design for a particularly complicated application, a multilevel graphical window system, which, unlike similar multilevel window system designs, can be implemented entirely in untrusted software. The straw man presented provides the user a multilevel view of system resources and allows the user to interact with applications executing at different security levels. Because no window system software need be trusted, this design is appropriate for high-assurance systems.

1 Background and Motivation

With the need for multilevel secure (MLS) trusted systems increasing, the natural problem of applying these systems to "real world" circumstances has arisen. A root of this problem is the lack of available applications that allow MLS system users the same flexibility they have become accustomed to on untrusted systems. Unfortunately, "applications for trusted systems" too often are turned into "*trusted* applications for trusted systems." In [23], Schaefer and Schell noted:

> The designers of a trusted system have two primary goals: the first is to reduce the size and complexity of the security kernel; the second is to produce a complete reference monitor implementation. When one is successful in meeting these goals, it is unnecessary to further consider the nature of untrusted code in the system.

Whereas the trusted system designer's goal is to completely and correctly implement the system security policy, the goal of the application designer should be to develop the application entirely in untrusted code. In general, the application designer should not need to consider how the application impacts system security, but rather how system security impacts the application.

A common misconception is that applications that provide users with a multilevel view of system resources must be trusted. To the contrary, an application that is sensitive to the underlying MLS environment can usually be implemented entirely with *untrusted code*. For example, the command interpreter in an MLS system is typically both aware of the MLS environment and untrusted. Such applications should:

- recognize and understand security labels, and use these labels to prevent users from attempting actions that will fail; and
- be designed to live within the security constraints typically enforced by current MLS products (e.g., the "no write down" constraints of the Bell-La Padula model's *-property [10, 11]).

Building untrusted applications to be aware of an MLS environment is a challenging, but practical task. The application designer must be cognizant of the underlying security policy and build the application accordingly [8, 15, 24].

Most existing application software was not developed with multilevel security in mind and often such applications have conventions that do not permit easy movement to an MLS environment. For example, a common practice is to use "well known" globally accessible files

*This paper reflects work completed in September 1990 under DARPA contract F29601-87-C-0071 while Mr. Mayer was affiliated with SPARTA, Inc.

or directories to store temporary information (e.g., the /tmp/ directory in UNIX[1]). In an MLS environment, if users at different security levels use the application (e.g., operating system utilities, text editors, spread sheets), it becomes impossible to use a single, well known object for this purpose (i.e., higher level subjects would otherwise be able to "write down"). Operating system design can avoid some such circumstances (e.g., see [16, 21]). However, in general, the application designer must use programming conventions to avoid potential conflicts with the system security policy.

In our experience, an increasingly common application design approach is to give a portion of the application software privilege with respect to the underlying trusted computing base (TCB)[2] in order to eliminate problems between the application software and the MLS security policy. While this approach expedites the implementation of the application, it has several negative consequences. By making application software trusted, the application developer has in effect modified the underlying TCB, and therefore potentially invalidated the assurances provided by any previous technical evaluation. In practice, some modifications may be demonstrably minor and manageable. However, modifications that change the basic flow of information (e.g., by allowing application code to "write down") are much more difficult to verify as safe. Even if the application developer is privy to all the design information, verification evidence, coding standards, and evaluation data available for the trusted system, the task of providing credible assurance that the application software does not violate the basic policy of the system is nontrivial. For example, Ames and Keeton-Williams noted that even when the trusted portion of the application is minimized to the absolute minimum necessary (in their case, a simple downgrade function), the task of showing correct security operation is surprisingly difficult [8].

If "trusted applications" were limited to those which absolutely required the ability to violate the system security policy (like downgrading), half of the application design problem would be solved. However, a tendency is to make application software trusted for reasons other than security relevance. Specifically, "convenience" and loosely defined "performance" reasons are often given for such design decisions. A "convenience" argument should rarely (if ever) be acceptable. This argument is typically asserted because the application designer lacks sufficient understanding or insight into an MLS programming environment (e.g., "it was simpler to make the application privileged and trust it not to violate the policy than to re-design the application to function within the system security constraints").

Performance issues are more difficult to assess. Given appropriate justification, performance may be sufficient rationale to include non-security relevant software in the trusted portion of the system. However, the justification must be great and the complexity and privilege of the trusted software limited [6, 7]. A suggested approach for building a strong "performance" argument is analogous to a traditional technique used to solve a similar performance issue with high-level languages, where a common practice is to initially develop an application entirely in a high-level language. Then, once the application is completely implemented, a few critical "bottle-neck" sections of the code are rewritten in a machine language. This technique provides substantial performance increases while maintaining nearly all the benefits of high-level languages. An analogous approach for application design on trusted systems is to implement the entire application with untrusted code and then determine if and where performance is a problem. Such an approach will achieve greater justification and limitation of the need (if any) for trusted code, just as the need for machine language can be limited without severe impacts on performance.

In this paper, we present a straw man design for a complex application for MLS trusted systems—in particular, an MLS window system. A window system application was chosen both because it is a good example of a complex application and because there is great need for such an application on high-assurance trusted systems.

2 Design Issues and Assumptions

The intent of this straw man is to present a design strategy for an MLS window system (WS) which can be implemented on a high-assurance trusted system (i.e., B3 or A1 according to [2]) without adverse impact on the assurance of the underlying TCB (i.e., with no or very little trusted WS code).

A modern WS is a particularly complicated set of application code supporting sophisticated graphic presentation formats, rich application interfaces, powerful user interfaces, inter-application communication,

[1] UNIX is a registered trademark of AT&T.

[2] A TCB is defined by [2] to be the "totality of protection mechanisms within a computer system." For high-assurance trusted systems (i.e., B3, A1), the TCB would primarily consist of an implementation of the reference monitor concept such as a security kernel (see [7, 9]).

management of complex devices (i.e., graphic terminals), device-independent programming protocols and conventions, and management of multiple application "windows." Because of this complexity, it is vital to ensure that the vast majority, if not all, of the WS software remains untrusted for at least the following reasons:

1. As previously noted, even the simplest addition of trusted code can invalidate the assurance of the underlying system, and certainly the addition of large amounts of complex trusted code is in opposition to a high-assurance design.

2. Since the WS software remains untrusted, the WS developer has great freedom to redesign, modify, maintain, and tune the WS software without concern for the protection of data. Given the complex nature of a WS, and the rapid developments occurring in WS standards, this development freedom is important for future enhancements and maintenance. Even in low (B1) or medium (B2) assurance systems, the ability to maintain, change, and evolve WS software without affecting the TCB may be compelling reasons to use the design approach suggested by this paper.

Using the X window system paradigm [20], a WS allows *clients* to interact with *servers* via an accepted protocol. Clients are applications that use a server's services to display data and to receive input from a graphic terminal. A server manages the terminal hardware, controlling both the screen display output, and keyboard (and mouse) input. Essentially, a server "owns" the terminal and allows applications to interact with the terminal through its services. The server provides clients with a standard, device-independent interface for windowing and graphic-display functions.

The MLS window systems that were currently being developed or proposed at the time this paper was written (e.g., see [4, 12, 13, 22]) all incorporate object labeling and access mediation within a large portion of the window system, typically at least the server, necessitating the inclusion of complex WS software within the TCB. Such an approach may be successful for low-assurance (B1) and possibly medium-assurance (B2) systems. However, for high-assurance systems, the introduction of large, complex WS function into the TCB is incompatible with the concept of a reference monitor and the B3 system architecture requirements [2, 7, 18, 22, 23]. In fact, none of the functions of a WS are inherently required for the enforcement of the security policy, and therefore would be difficult to argue as necessary for inclusion within the TCB as suggested in [7, 23].

The straw man presented herein provides a multilevel secure user interface via a graphic workstation, with a minimum of trusted code. In fact, as will be discussed, no code inherent to the WS need be trusted.

This MLS WS design exploits the natural communication paths provided by systems that support a Bell-La Padula style security policy. In such systems, subjects are allowed to read objects at security levels dominated by (are lower than) the subject's security level and to modify objects at security levels that dominate (are higher than) the subject's security level[3]. These capabilities allow coordination between subjects of differing security levels only in accordance with the system security policy (i.e., information may "flow up", but not "down"). Such capabilities are central to the success of this design.

3 Device Management Issues

The MLS WS design expects the underlying TCB to treat the graphic terminal in a rather non-traditional manner. It is common for trusted systems to implement a terminal device as one, single-level-at-a-time device. Any application accessing the device may do so only in accordance with the mediation rules applied for all subject-object accesses. Some trusted systems allow users to change the current security level of the terminal, via a trusted path, without re-authentication (e.g., SCOMP [14] and GEMSOS [25]). However, the terminal still remains single-level at any given moment.

For the MLS WS design, it is essential that the reference monitor treat the terminal as *two* separate and distinct devices—namely an output device (the screen) and an input device (the keyboard and mouse). While most current MLS systems do not treat terminals in this manner, it is a natural approach, especially for

[3] "Dominates" is defined in [10, 11]. While the Bell-La Padula models allow these functions, they do not require them. Most systems provide "read-down" or "write-up" capabilities for some objects, but not all. There is in fact very little distinction between the two forms of access—one form can emulate the other (e.g., "write-up" can be emulated by a low-level subject writing an object at its level and a high-level subject "reading down" to the low-level object). In practice, most Bell-La Padula style systems provide at least "read-down" capabilities for "large" objects such as memory segments or files. The availability of "read-down" and "write-up" capabilities for a given system can have a significant performance impact on this design.

console devices where the screen and keyboard are typically two separate physical devices with separate communications ports to the system. GEMSOS, for example, implements input and output (and their associate control functions) as distinct devices [5]. The MLS WS design also requires the TCB to provide a means for the user to change these device levels via a trusted path (although they will remain single-level devices at any given time), much the same as allowed by SCOMP and GEMSOS. In the remainder of this paper, we will refer to the current security level of the keyboard and other input devices (e.g., a mouse) as the *terminal input level* and the current security level of the screen as the *terminal output level*.

The ability to separate input and output as distinct objects, and to change the device current security levels, are the only trusted features necessary to support the MLS WS design. Neither of these features are intrinsic to a WS and can be provided by a trusted system as part of its basic terminal management functions.

4 Server Design

To be useful, this design must provide as much compatibility as possible (and still meet the high-assurance goal) with application code written for a standard window system (e.g., X). Therefore, changes to the client side of the WS protocol must be minimized (if not completely eliminated). In the MLS WS design, the MLS functionality is provided exclusively by the design of an untrusted server that is aware of the underlying MLS environment. As previously noted, a WS server provides graphic-terminal input and output management services via a standardized protocol. Typically, this server is implemented as a single monolithic subject (e.g., a process in Unix [20]) that performs all the desired tasks. In the MLS WS design, the server is divided into two major functional areas which are distributed among a number of single-level untrusted subjects (see Figure 1). These server subjects, which are called *session managers* and *screen managers*, are described below.

4.1 Session Managers

Session managers have three primary responsibilities:

- service client requests;
- distribute terminal input; and
- manage *window buffers*.

Session managers are single-level, untrusted subjects. A client subject interacts with a session manager running at its security level. This allows clients and servers (i.e., the session manager) to participate in active, two-way communication. If clients at differing security levels wish to interact with the same terminal, then it is necessary to have a session manager *active* at each client security level. It is not, however, necessary to have session managers at *all possible* security levels. Rather, when the user changes the terminal input level, a new session manager can be created (if one does not already exist at that security level). Given the necessary system support for changing terminal input level, the creation of a new session manager can be handled entirely by convention via a traditional login initialization procedure (e.g., via a ".login" file for the Unix C-shell). Therefore, a session manager will be active at each security level for which the user has active client subjects.

Terminal input is directed to the session manager running at the current terminal input level. This session manager decides how to distribute terminal input. For example, input can be forwarded to a client, sent to a window within the window buffer, forwarded to the screen manager, or interpreted as a session manager command from the user. The only input not directed to the session manger is the "secure attention key," which of course the underlying TCB recognizes as a request to invoke the trusted path.

Window buffers are single-level "views" of the physical terminal screen. Every session manger has an associated window buffer which it manages. Client requests to manipulate the terminal screen (e.g., create a window, resize a window, input data to a window) are reflected to the window buffer. Abstractly, a window buffer can be viewed as a single object, but in reality, any number of objects may be used to implement the buffer (e.g., a combination of shared memory regions, IPC channels, and files). All clients at the same security level will have their windows managed by the same session manager in the same window buffer.

4.2 Screen Manager

Whereas a session manager controls and distributes terminal *input*, the screen manager is responsible for terminal *output*. Like the session managers, the screen manager is a single-level, untrusted subject. However, unlike session managers, there is only one screen manager per terminal. The screen manager's basic function is to read all window buffers and display their contents

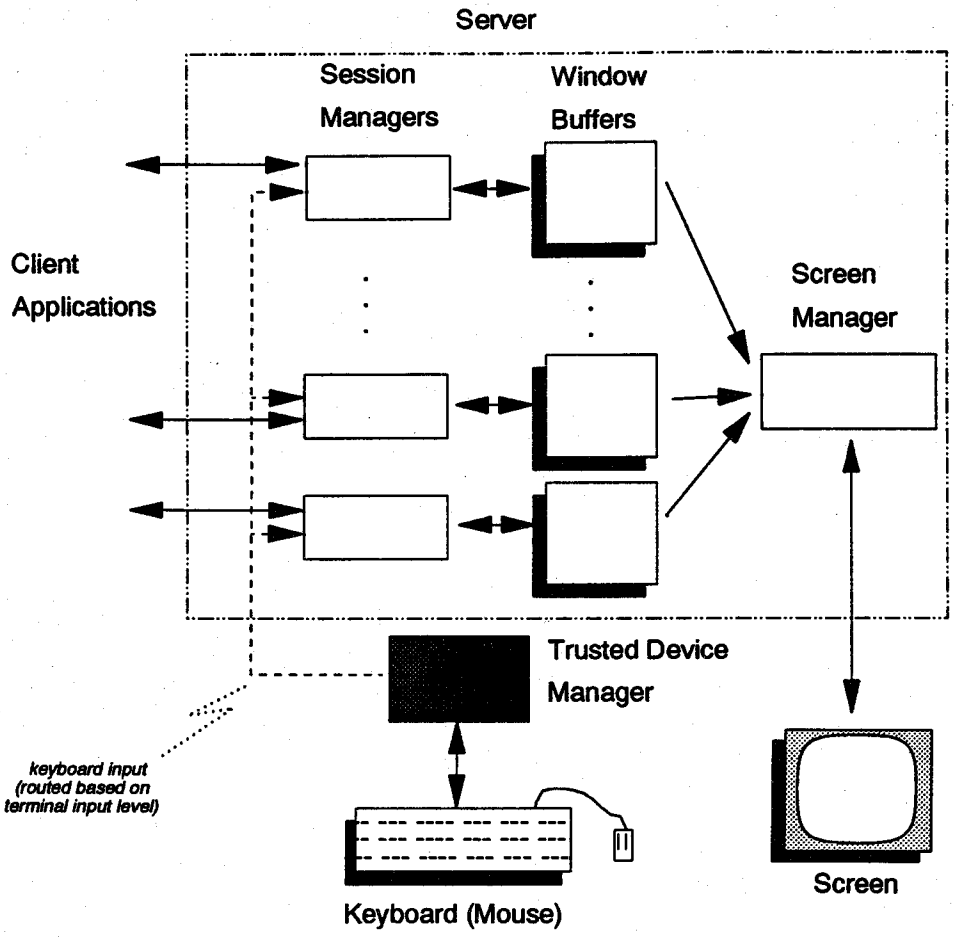


Figure 1: Overview of the MLS Window System Design

on the terminal's graphic screen. The intra-server protocol between session managers and the screen manager must be sufficiently descriptive to facilitate graphic display operations. The screen manager decides the order and placement of windows displayed on the screen, based upon information provided by the session managers (window dimensions, resizing, expose or hide windows). While the screen manager controls the physical terminal screen, it does not necessarily service all apparent screen functions. For example, a client request to "read" the contents of a given location on the screen may actually be serviced by a *session* manager, which would return the requested information from its window buffer and not the physical screen.

Since the screen manager is a single-level untrusted subject that manipulates the screen, it must run at the terminal output level. In the simple case, the terminal output level (and therefore the screen manager level) would be *session high* (i.e., the highest level the user is allowed to use at that terminal). This would allow the screen manager to read all possible window buffers ("read-down" is allowed). Thus, one screen manager can service all session managers running at separate security levels without the necessity of trusted code.

In a more general design, the user can change the current terminal output level and therefore the current screen manager level. This would require the screen manager to be restarted; essentially creating a new single-level subject as the screen manager at the new terminal output level. Of course, a screen manager can only read the window buffers at its level or a lower level. So if there exists session managers at a higher security level than the current terminal output level, the screen manager will not be able to display those windows.

The advantage of allowing the screen manager to be restarted at a new security level is that users can limit

their view of the system resources. For example, if a user, whose current screen manager is running at Top Secret, wished to ensure that the display contained only Secret information, the user would change the terminal output level to Secret resulting in a new screen manager at Secret. Since the screen manager is untrusted, the user is guaranteed that any information displayed is no higher than Secret. In an ideal design, multiple screen managers would co-exist much the same as multiple session managers co-exist. In this scenario, the screen manager executing at the current terminal output level would have control of the screen. This would allow the user to switch the screen level without the overhead of restarting a new screen manager.

The disadvantage of switching screen managers or allowing multiple screen managers is that the interaction between session managers and screen managers is complicated. An initial design approach would be to implement a fixed, session high screen server and work incrementally towards more dynamic screen management. The fixed, session high screen manager approach will require the user to re-initiate the session (i.e., logout and login) in order to change the terminal output level. Nonetheless, a single, session high screen server will still allow the user to concurrently display and manipulate windows at multiple security levels.

5 Clients and User Interactions

The above server design requires some communication between the screen manager and session managers in order to properly coordinate display operations. To illustrate, assume the terminal output level is currently Top Secret and the terminal input level is Secret. In the course of operation it is possible for a Secret client to create and display information in a manner that the user did not intend. An example may be a graphics drawing program where the user's keyboard input causes the client to display graphic images. The client "displays" these images by sending requests to the Secret session manager which translates the client's requests into screen manager instructions and places these instruction in the Secret window buffer. The Top Secret screen manager would then display the new contents of the client's window. An input mistake would be recognized by the user only after the image is displayed by the screen manager. Now we have a situation where a Secret subject (i.e., the drawing client) must be given new instructions (i.e., re-draw the image) based upon information from a Top Secret subject (i.e, the screen manager)—see Figure 2. This would seemingly be a violation of a Bell-La Padula style security policy model (i.e., a "write-down").

However, the recognition and correction of the drawing error is performed by the ***human user***. The user sees the mistake displayed on the screen and types new instructions to correct the error. The user's keyboard input (which recall is Secret) is directed by the Secret session manger to the Secret client drawing program, which would once again send the appropriate requests to the Secret session manager to cause the new image to be displayed.

This example illustrates a more general observation about an MLS window system. The "downward" feedback (i.e., the coordination between the screen manager display of a window and the client subject which owns the window) usually occurs via the human user outside of the computer. This is an important observation. Since client, session manager, and screen manager subjects are all single-level untrusted subjects, an underlying trusted system will not allow this "downward" feedback to occur without the inclusion of special trusted code (which this design is trying to avoid). However, since "downward" feedback occurs naturally via the human user, no violation of the system security policy exists and no trusted code is necessary.

Another simple example of "downward" feedback is exposing or hiding windows. For example, in the above scenario, the screen manager could currently have a Top Secret window displayed "on top of" a Secret window that the user wishes to view. In order to expose the Secret window, the user would input the appropriate command directing the Secret client and session manager to instruct the Top Secret screen manager to expose the Secret window.

It appears that the vast majority (if not all) actions requiring this "downward" information flow occurs naturally via the human user interface, thereby avoiding the need for trusted application software.

Another common feature that a WS provides is a ***cut and paste*** function. Abstractly, a cut and paste function allows information in one window to be copied into a ***cut buffer*** which then may be imported to another window. In an MLS window system, it is desirable to allow cut and paste functions between windows at differing security levels. In our design, this function can be implemented in a number of ways. For example, to cut a section from an Unclassified window and paste it to a Secret window, the user must first ensure that the terminal input level is set to Unclassified to allow manipulation of an Unclassified window (see Figure 3).

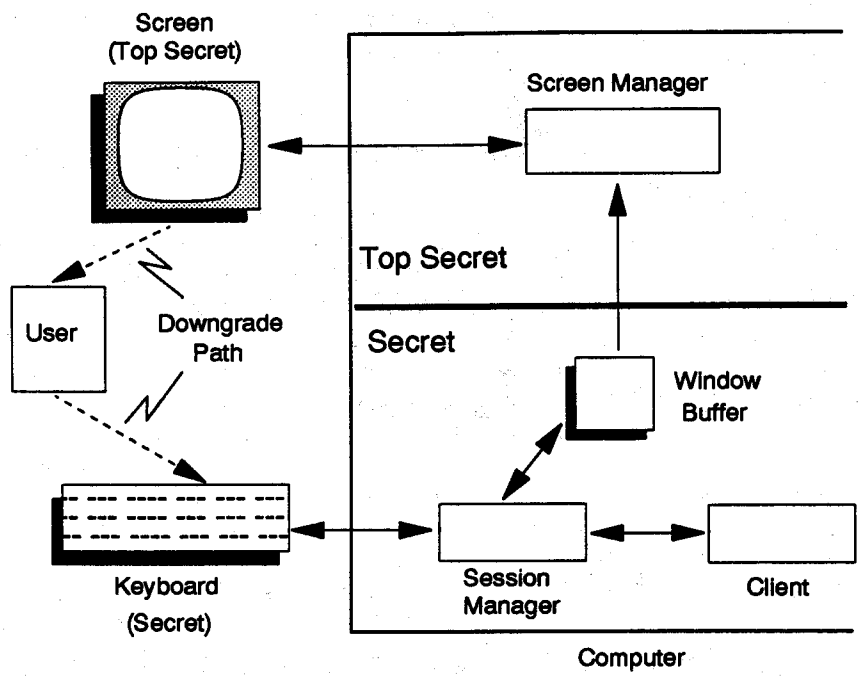


Figure 2: Example: "Downward" Feedback Occurs via the User Outside of the Computer

The user can then select the appropriate window region and request a "cut" operation. The client that owns the window would ensure that the selected region is copied into the Unclassified cut buffer. The cut buffer could be implemented in a number of ways, but for this example assume that it is a "well known" file object. The key point is that the cut buffer is an object and therefore must have a single security level (i.e., Unclassified in this case). To fully accommodate a cut and paste function, a cut buffer must exist for each active security level (i.e., one for each session manager).

In order to "paste" the information in the Unclassified cut buffer to a Secret window, the user must change the terminal input level to Secret (which allows manipulation of Secret windows). The user can then issue a conventional "paste" operation with one possible exception; the user may be required to specify the security level of the cut buffer from which to paste—in this case Unclassified (otherwise the Secret client would likely use the Secret cut buffer by default). Since a Secret subject is permitted to read an Unclassified object, the paste operation will succeed.

The actual implementation of a cut and paste function will likely be more complex and innovative than the above example. For instance, the "cut" operation in the above example can be performed by the Secret client "reading down" to the Unclassified window buffer, avoiding the need for the user to change the terminal input level in the middle of the cut and paste function. Also, pull down menus for specifying cut buffer security levels can moderate any undesirable user interface impact. The important concept is that operations, like cut and paste, which are in accordance with the system security policy, can (and should) be implemented by programming convention, without trusted code.

Notice that the above cut and past example does not allow "downward" pasting (e.g., cutting from a Secret window and pasting to an Unclassified window). Such a flow is in violation of the system security policy. This constraint leads to a natural question: Since a cleared individual is trusted to properly review and sanitize information, why not allow the individual to do so via an automated cut and paste function? The answer to this question is apparent in the basic distinction between [untrusted] software and human users. In a computer, information can be masked or hidden from the user by illicit software (i.e., a Trojan horse). Even in very simple forms (i.e., ASCII text), seemingly harmless information may contain illicit data which is difficult, if not

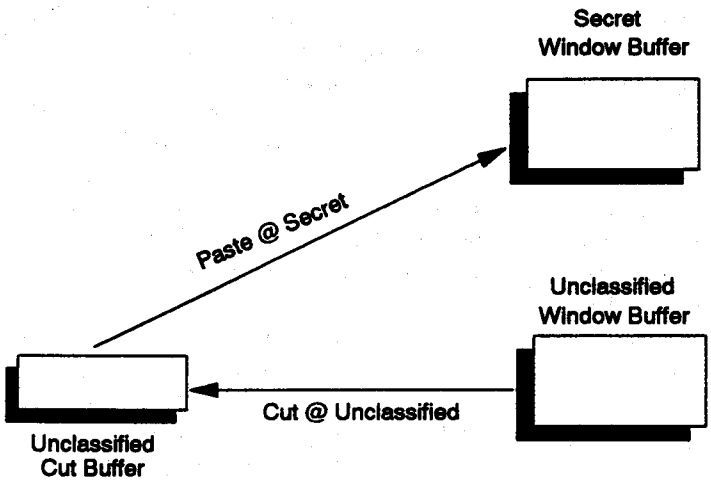


Figure 3: Example: Cut and Paste Across Security Levels

impossible, for a human to detect [19]. To illustrate, examine the following paragraph:

```
Information is easy to hide from a human being
within formatted text.  Take for example this
simple paragraph.  From a potential reviewer's
perspective, this text is harmless unclassified
material. However, buried within the formatting
of this text is a secret message that can
be easily decoded by a computer but is nearly
impossible for a reviewer to recognize.  The
content of this paragraph certainly does not
convey the secret message by any overt means.
However, the right justified format has the
message encoded within it.
```

Other than the fact that the paragraph states that an illicit message is encoded within the text format, it is very unlikely a typical user would recognize the existence of hidden information (see the Appendix for encoding scheme). A trusted downgrader that simply displayed information to the user and then allowed the information to be downgraded would be highly susceptible to this type of an attack, especially if the user often downgrades data.

In McHugh's design of a high-assurance downgrader [19], the type of encoding scheme used in the above example is addressed by removing any extra spaces (and therefore any special formatting). However, McHugh noted that other forms of encoding may exist and ultimately the user must assert that no illicit encoding exists. McHugh's and other similar high-assurance downgrader designs [8, 17] exhibit several common characteristics:

- only simple forms of data (e.g., ASCII text) are handled;
- the "trusted" portion of the downgrader is very simple and therefore verifiable; and
- the user interface is necessarily cumbersome to force the user to carefully scrutinize the data before downgrading.

Although implementing a downgrade cut and paste function in a window system is conceptually similar to these other designs, such a mechanism has at least the following significant differences:

- the trusted portion of the downgrader must include all software capable of influencing the displayed text (which would likely include server software and some clients), leading back to the situation where complex software must be trusted and thereby precluding a high assurance design; and
- the ability to downgrade sophisticated data forms (e.g., graphics, special fonts) greatly increases the ability of illicit code to hide classified information in seemingly unclassified data (e.g., subtle changes in font sizes, slight pixel adjustments in graphics).

These differences make an intuitively safe operation (from the user's perspective) impractical to implement

in a window system given the current state-of-the-art for high-assurance trusted systems.

Nothing in the MLS WS design prevents a user from intentional sanitizing information through re-keying the data. For example, assume the terminal output level is Top Secret and the terminal input level is Secret. The user can view Top Secret information on the terminal screen and type in the sanitized information to a Secret window. However, in this situation the burden for ensuring that only appropriate information is downgraded is placed entirely on the user. The WS gives the user no guarantee of the accuracy of the data displayed other than the terminal output and input levels.

6 User Visible Labels

Ultimately, the issue of labeling windows must be addressed. We argue that no security labeling of windows is required for this design. The reason is simple—windows are abstractions created by untrusted software out of TCB provided objects, and the TCB will ensure that the underlying objects are properly labeled and protected.

The objects implemented by the TCB in this design are the terminal input and output objects—nothing more or less. Windows are abstractions created by the untrusted Screen Manager, and as such are not abstractions known to the TCB. In essence, windows are analogous to abstractions created by any untrusted application (e.g., forum meetings in Multics, a spread sheet). It was never the intent that a high-assurance TCB should understand and manage all abstractions in a system [23, 9, 7]. Rather, a well-designed TCB protects system resources by grouping them into primitive abstractions called objects. The untrusted portions of the operating system and applications software can create more complex abstractions such as windows out of these TCB objects. Nonetheless, no matter what abstractions are created by untrusted software, the fundamental protection of system resources provided by the TCB will not be changed (i.e., the security policy cannot be violated by untrusted software).

The concept of implementing windows by non-TCB software is unlike other approaches typified by the recent compartmented mode workstation effort at Mitre [22, 26, 27], which incorporates the window system software into the TCB.

A common argument for having the TCB label windows is that data becomes overclassified and must eventually be "downgraded" to its original security level. Another argument is that without reliable window labels, the user can accidentally confuse sensitive information with non-sensitive information. Because of these concerns, it can be argued that trusted window labeling, coupled with a trusted "downward" cut and paste function, is needed. These arguments, however, lead back to the situation of trusting large portions of the WS software, both to reliably label information and to provide the appropriate guarantees for the "downward" cut and paste function.

As noted previously, complex WS software in the TCB is incompatible with a high-assurance design, and certainly with a potentially dangerous "downgrade" function. Since, in the MLS WS design, the terminal's output and input levels can differ, the user can avoid data overclassification while still having a multilevel view of the system resources. In a typical MLS system, users must login at the level of the most sensitive data they wish to view; resulting in any newly created objects being labeled with that high level. In the MLS WS design, the user can view objects (through windows) at any level at or below the current terminal output level, while entering new data at the (potentially lower) terminal input level. While the screen manager may display object security levels on windows, any labels, other than the terminal input and output levels, are unreliable.

Potential criticism of this approach is that the untrusted screen manager may attempt to spoof the user into unintentionally downgrading information through mislabelling of windows. The response to this criticism is simple. Any time an individual chooses to downgrade information, great care must be taken to ensure that the information being downgraded is clearly not classified. As the example in the Appendix illustrates, downgrading is technically unsafe with or without TCB-provides labels. Downgrading should be the *exception*, not the rule! The situation where the screen manager prints bogus security levels is no different than application code printing bogus labels on printed output in a System High or Dedicated mode system. Ultimately, the user must ensure that the data are appropriate for downgrade based on its *content* and not on any outwardly visible labels on windows. In the MLS WS design, the user can always limit the level of information on the screen by changing the terminal output level (and therefore the screen manager level) to a lower security level.

In the case of a "trusted downgrade" cut-and-past function, a prominent consideration is the reliability of the software capable of violating the system secu-

rity policy (including that responsible for displaying the data to the user). As discussed in the previous section, high assurance cannot be obtained for large amounts of complex software like a WS terminal server. Depending on such software to be reliable and "leak proof" may be more dangerous than the benefit achieved. Such concerns are indeed the basis for the greater risks a higher assurance system can manage versus a lower assurance system [1, 3]. As more MLS technology (e.g., local and wide-area networks, file servers, workstations, databases) is proliferated, the problem of over classified data, which is common in System High environments, will dissipate, mitigating the need for downgrade functions.

The MLS WS straw man does leave one labeling issue open—a means by which the user can determine the current terminal input and output levels (and to allow reliable communication when the trusted path is invoked). This is not a simple problem to resolve. An straightforward solution would be to "yank" control of the screen from the screen manager, re-initialize the screen, and give the trusted path software control of the screen. This approach is awkward and expensive, especially when the output device is complex and graphics are involved. A more complicated solution would be to virtualize the screen, saving a small portion of the display for the TCB to display security labels. However, this approach has the same shortcomings discussed previously (i.e., requires complex, trusted screen management). Ultimately, the solution may be for graphic terminals to provide a separate, simple output device to allow TCB-to-user communication. An example may be a small LCD display on the keyboard or some other separate, simple, and distinct display. Such a display would be used for TCB-to-user communication during invocation of the trusted path without interrupting the user's normal display. Other times, this display would show the current terminal input and output security levels so that the user is constantly aware of the terminal's current security levels.

This issue is one of many to be addressed in a more detailed examination of this straw man design.

7 Conclusions

The objective of this straw man is to demonstrate that complicated, multilevel secure applications can be built on trusted systems *without* complex trusted application software. While several open issues remain, the straw man design for an MLS window system presented appears to be feasible. The advantages of an entirely untrusted MLS window system are great. The window system software can be maintained, updated, and modify without concern for the underlying system security policy. New and better protocols and conventions can be introduced as they evolve. And probably the best advantage is that, by excluding complicated window system software from the trusted portion of the system, an MLS window system can be implemented on high assurance systems. Undoubtedly many issues remain to be addressed in the next design stage of the MLS window system proposed (e.g., performance, visible labels). However, an approach for developing complicated applications for trusted systems, similar to the straw man design described herein, can minimize, if not eliminate, the need for trusted application software.

References

[1] Computer Security Requirements. CSC-STD-003-85, National Computer Security Center, Fort Meade, MD, June 1985.

[2] *Department of Defense Trusted Computer System Evaluation Critera.* December 1985.

[3] Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements. CSC-STD-004-85, National Computer Security Center, Fort Meade, MD, June 1985.

[4] Final Evauation Report of AT&T System V/MLS. CSC-EPL-89/003, National Computer Security Center, Fort Meade, MD, 18 October 1989.

[5] Private communication between Steve Padilla and Albert Toa, September 1990. Unpublished E-mail Coorespondance.

[6] S.A. Ames Jr. Security Kernels: A Solution or a Problem? In *Proc. 1981 IEEE Symposium on Security and Privacy*, pages 141–150, Oakland, CA, April 1981. IEEE.

[7] S.A Ames Jr., M. Gasser, and R.R. Schell. Security Kernel Design and Implementation. *IEEE Computer*, pages 14–22, July 1983.

[8] S.A. Ames Jr. and J.G. Keeton-Williams. Demonstrating Security for Trusted Applications on a Security Kernel Base. In *Proc. 1980 IEEE Symposium on Security and Privacy*, pages 145–156, Oakland, CA, April 1980.

[9] James P. Anderson. Computer Security Technology Planning Study. Technical Report ESD-TR-51, vols. I and II, Electronic Systems Division, Bedford, Mass., October 1972.

[10] D.E. Bell. Secure Computer Systems: A Refinement of the Mathematical Model. Technical Report MTR-2547, vol. III, Mitre, Bedford, Mass., December 28, 1973.

[11] D.E. Bell and L.J. La Padula. Secure Computer System: Unified Exposition and Multics Interpretation. Technical Report MTR-2997, Mitre, Bedford, Mass., July 1975.

[12] M.E. Carson and et al. Fron B2 to CMW: Building a Compartmented Mode Workstation on a Secure Xenix Base. In *Proc. Third Aerospace Computer Security Conf.*, pages 35–43, Orlando, Forida, December 1987.

[13] P.T. Cummings and et al. Compartmented Mode Workstation: Results Through Prototyping. In *Proc. 1987 IEEE Symposium on Security and Privacy*, pages 2–12, Oakland, CA, April 1987.

[14] L.J. Fraim. Scomp: A Solution to the Multilevel Security Problem. *IEEE Computer*, pages 26–34, July 1983.

[15] D. Gambel and S. Walter. Retrofitting and Developing Applications for a Trusted Computing Base. In *Proc. 11th National Computer Security Conf.*, pages 344–346, Baltimore, Maryland, October 1988.

[16] V.D. Gligor and et al. On the Design and the Implementation of Secure Xenix Workstations. In *Proc. 1986 IEEE Symposium on Security and Privacy*, pages 102–117, Oakland, CA, April 1986.

[17] T. Hinke, J. Althouse, and R.A. Kemmerer. SDC Secure Release Terminal Project. In *Proc. 1983 IEEE Symposium on Security and Privacy*, pages 113–119, Oakland, CA, April 1983.

[18] C.E. Irvine and et al. Genesis of a Secure Application: A Multilevel Secre Message Preparation Workstation Demonstration. In *Proc. Fourth Aeorspace Computer Security Applications Conf.*, pages 30–36, Orlando, Florida, December 1988.

[19] John McHugh. An EMACS-Based Downgrader for SAT. In *Proc. 8th National Computer Security Conf.*, pages 113–136, Gaithersburg, Maryland, September–October 1985.

[20] A. Nye, editor. *X Protocol Reference Manual.* O'Reilly & Associates, Sebastapol, CA, 1990.

[21] E.I. Organick. *The Multics System.* The MIT Press, Cambridge, Mass., 1972.

[22] J. Picciotto. Trusted X Window System, Volume 1: Design Overview. Technical Report MTP-288, vol. I, Mitre, Bedford, Mass., February, 1990.

[23] M. Schaefer and R.R. Schell. Toward an Understanding of Extensible Architectures for Evaluated Trusted Computer System Products. In *Proc. 1984 IEEE Symposium on Secuirty and Privacy*, pages 41–49, Oakland, CA, April 1984.

[24] E.R. Schallernmuller, R.P. Cramer, and B.T. Aldridge. Development of a Multilevel Data Generation Application for GEMSOS. In *Proc. Fifth Computer Security Applications Conf.*, pages 86–90, Tucson, Arizona, December 1989.

[25] W.R. Shockly and D.F. Warren. Description of Multilevel Secure Entity-Relationship DBMS Demonstration. In *Proc. 11th National Computer Security Conf.*, pages 17–20, Baltimore, Maryland, October 1988.

[26] J.P.L. Woodward. Exploiting the Dual Nature of Sensitivity Labels. In *Proc. 1987 IEEE Symposium on Security and Privacy*, pages 23–30, Oakland, CA, April 1987.

[27] J.P.L. Woodward. Security Requirements for System High and Compartmented Mode Workstations. Technical Report MTR-9992, Rev. 1, Mitre, Bedford, Mass., November 1987.

Appendix

The hidden message in the following paragraph is the word "COVERT."

```
Information is easy to hide from a human being
within formatted text.  Take for example this
simple paragraph.  From a potential reviewer's
perspective, this text is harmless unclassified
material. However, buried within the formatting
of this text is a secret message that can
be easily decoded by a computer but is nearly
impossible for a reviewer to recognize.  The
content of this paragraph certainly does not
convey the secret message by any overt means.
However, the right justified format has the
message encoded within it.
```

The algorithm for encoding bits in the text is as follows:

> Examine all words that contain the letter "t", ignoring all words that begin either a line or a sentence. If a selected word has greater than one space before it, a binary "1" is signaled. If there is exactly one space before the word, a binary "0" is signaled.

Information is represented in bits via the following trivial encoding scheme:

> Assign each letter of the alphabet an ordinal value such that A=1, B=2,...,Z=26. Assign a blank space the value 0. Each letter is then encoded in a 5 bit binary number. These binary numbers are concatenated into a bit stream and hidden in the text with the above algorithm.

The paragraph is repeated below with the encoded binary digits explicitly included (in place of a space).

```
Information is easy0to hide from a human being
within0formatted0text.  Take for example 1this
simple paragraph.  From a 1potential reviewer's
perspective,0this 1text is harmless unclassified
material. However, buried 1within 1the 1formatting
of 1this0text is a 1secret message 1that can
be easily decoded by a0computer0but is nearly
impossible for a reviewer0to recognize.  The
content of 1this paragraph0certainly does 1not
convey 1the0secret message by any0overt means.
However, 1the0right 1justified0format has 1the
message encoded0within0it.
```

Remembering that each letter is encoded in a 5 bit number, we should get the following message:

```
00011 =  3 = C
01111 = 15 = O
10110 = 22 = V
00101 =  5 = E
10010 = 18 = R
10100 = 20 = T
```

This is a very simple encoding scheme. More sophisticated schemes exist that will provide much greater bandwidth.

EXPERIENCE WITH A PENETRATION ANALYSIS METHOD AND TOOL

Sarbari Gupta Virgil D. Gligor

Electrical Engineering Department
University of Maryland
College Park, Maryland 20742

ABSTRACT

We present a penetration-analysis method, an experimental tool to support it, and the experience gained from applying this method and tool to the Secure Xenix℠ source code. We also present several properties of penetration resistance, and illustrate their interpretation in Secure Xenix using several penetration experiments. We argue that the properties of reference monitor mechanisms are necessary but insufficient to provide penetration resistance for a system. However, the assurance process for establishing penetration resistance need not differ from that required for demonstrating support for access control policies.

1. INTRODUCTION

The penetration-resistance of a computer system is a separate security concern from that of supporting access control and accountability policies. Different systems may exhibit the same degree of penetration resistance, but implement widely different access control or accountability policies, or may implement the same policies, but exhibit different degrees of penetration resistance. Furthermore, not only the effectiveness of these policies, but that of other policies, such as that of system availability, depends on the penetration-resistance of a system.

Despite the obvious importance of a system's penetration resistance, general methods for systematic penetration analysis have not been available. The first attempt at such analysis, the Flaw Hypothesis Methodology [11], consists of the generation of "flaw hypotheses," via largely *ad hoc* means, and the confirmation of these hypotheses, via system tests. The Flaw Hypothesis Methodology provides neither a systematic way of deriving flaw hypotheses nor a means to establish penetration-test coverage. Using this methodology, one cannot formally verify any penetration-resistance properties.

In this paper, we present a new penetration-analysis method, which (1) provides a systematic approach to penetration analysis, (2) enables the verification of penetration-resistance properties, and (3) is amenable to automation. We also describe an experimental tool, called the Automated Penetration Analysis (APA) tool, to support the penetration analysis method, and present the experience gained from applying this method and the tool to Secure Xenix [6] source code. We illustrate several properties of penetration resistance in the context of penetration experiments performed on Secure Xenix.

Our experience with this penetration-analysis method and experimental tool leads to three general observations. First, the penetration resistance of a computer system may not rely exclusively on the penetration resistance of its Reference Monitor Mechanism [1,12], contrary to long-standing belief. Penetration resistance includes additional properties, which differ from those of the Reference Monitor Mechanism, and expands the scope of the existing Reference Monitor properties of isolation and noncircumventability. Thus, penetration analysis gains *added significance* in the design, implementation, and verification of secure systems beyond that of the Reference Monitor properties.

Second, the ability to verify penetration resistance using source code depends on the ability to correctly derive validation-check specifications – which are required to determine the correctness of source code – from abstract penetration-resistance properties. This derivation is *dependent on the design and programming disciplines* used in system development. For example, parameter validation checks may depend on the semantics of the system call, object type, and parameter type used; validation checks for trusted processes may depend on the privilege acquisition and inheritance disciplines used. These dependencies suggest that the use of certain design and programming disciplines may in fact aid the analysis of penetration-resistant systems.

Third, the penetration analysis of a computer system can follow a *similar assurance process* as that for access-control and accountability policies. For example, (1) the penetration-resistance properties must be interpreted in the internal architecture specifications of a system in an analogous manner to that used to interpret the policy models in the top-level specifications of a

℠ Xenix is a trademark of Microsoft, Inc. Secure Xenix is an early version of Trusted Xenix, a product of Trusted Information Systems, Inc.

system [12]; (2) the specification-to-code correspondence must be performed to show that the penetration-resistance properties are preserved by source code in an analogous manner to that used to show that access-control invariants are preserved by source code [14]; and (3) penetration testing must be performed to show that the penetration-resistance properties are preserved by object code produced by a (trusted) translator. Thus, the degree of assurance required for penetration resistance need not differ from that for access control and accountability policies.

The remainder of the paper is organized as follows. In Section 2, we review the theory of penetration-resistant systems. In Section 3, we present the Penetration Analysis Method. In Section 4, we describe the Automated Penetration Analysis tool structure, illustrate the use of the tool using a penetration example, and present the experiments conducted using the tool. In Section 5, we discuss the insights gained from the experiments. Finally, in Section 6, we conclude this paper by suggesting directions of future research.

2. A THEORY OF PENETRATION-RESISTANT SYSTEMS

In this section we review the theory of penetration-resistant systems presented in reference [7](*). We define *penetration* as a method of exploiting system flaws to gain illegal or unintended access to system variables, objects and/or operations. The notion of access to variables refers to either *viewing* or *altering* access and access to operations refers to the capability to *invoke* a command or system-internal function. The terms "illegal" and "unintended" refer to accesses that violate one or more properties of penetration resistance. Our definition of penetration does not address illegal accesses obtained via operational security errors. It only considers errors in the source code, but not of hardware, that may cause vulnerability to partial or complete subversion of the security controls of the system by untrusted user processes and commands.

The penetration-analysis method is based on a theory of penetration-resistant computer systems, a model of penetration analysis, and a unified representation of penetration patterns [7]. The theory consists of the Hypothesis of Penetration-Resistant Systems and a set of design properties that characterize resistance to penetration. The penetration-analysis model defines a set of states, a state-invariant for penetration resistance, and a set of rules that can be applied for analyzing the penetration vulnerability of a system. An interpretation of the Hypothesis of Penetration-Resistant Systems within a given system provides the Hypothesis of Penetration Patterns, which enables us to define a unified representation for a large set of penetration instances as missing check patterns.

The Hypothesis of Penetration-Resistant Systems states that a system (e.g., a TCB) is largely resistant to penetration if it adheres to a specific set of design properties. The set of design properties, which are called the penetration-resistance properties, include:

- System Isolation (or Tamperproofness) – ensures that the system is isolated (or protected from tampering) from untrusted user processes. It involves system call parameter validation, system/user address space separation checks and control of system entry points (e.g., system privilege checks).
- System Noncircumventability – guarantees that all object references are mediated by the access check modules within the system. Object references include references to object contents, object status variables, object privileges and other subjects.
- Consistency of System Global Variables and Objects – maintains the invariant assertions that hold over the global variables, objects and internal functions of the system.
- Timing Consistency of Condition (Validation) Checks – assures that the validity of a condition (validation) check is not lost at the moment when an action that depends on that check is actually performed [3].
- Elimination of Undesirable System/User Dependencies – ensures that unnecessary dependencies between system and user are not present in the system [5].

The penetration-resistance properties are captured in the penetration-analysis model by the model constants and the state-transition rules [7]. The model is a state-transition model based on the policy that a system entity may be *altered* or *viewed*, or a system internal function may be *invoked*, only if the *set of conditions* associated with the alter/view/invoke access specified by penetration-resistance properties are validated in an atomic sequence (with the alter/view/invoke operation itself.) The model defines a system state as the set of integrated flow paths traversed by the system up to a certain point in time, a state invariant for penetration resistance, and a set of state transition rules that define secure state transforms.

The Hypothesis of Penetration Patterns suggests that system flaws, which are caused by incorrect implementation of the penetration-resistance properties, can be identified in system (e.g., TCB) source code as patterns of incorrect/absent validation-check statements or integrated flows that violate the intended design or code specifications.

(*) An extensive set of references to penetration attempts in operating systems is included in [7].

To represent penetration patterns uniformly, we define the notion of the *integrated (execution) flow path* within a system call, which consists of (1) the information flows, (2) the function calls, and (3) all the conditions checked along the execution path. An integrated flow path starts at an entry point of a system call and follows through various internal functions via call/return statements until the execution of the system call concludes. [In this paper, we will interchangeably refer to an integrated (execution) flow path as an integrated path, an integrated flow, or a flow path.]

3. THE PENETRATION ANALYSIS METHOD

We use the theory of penetration-resistant systems to derive the penetration-analysis method. The Hypothesis of Penetration Resistant-Systems tells us that a system becomes vulnerable to penetration attacks if the penetration-resistance properties are improperly implemented by the system. These properties are used to derive the validations (or condition checks) required to be done prior to the occurrence of various information and function flows within the system or TCB.

The Hypothesis of Penetration Patterns indicates that these required validations should be present in all execution paths within the system. Thus, the integrated flow paths provide a way to represent execution paths in a format which may be analyzed for the presence/absence of the required validations. The model supplies us with a set of rules to systematically analyze the integrated execution paths for the presence of the required checks. Thus, the penetration-analysis method is a judicious application of the theory of penetration-resistant systems [7].

The penetration analysis method consists of three stages:

Stage 1: execution path integration. In this stage, all the integrated (execution) flow paths for the system under consideration are generated using (information and control) flow based tools on the system source code.

Stage 2: derivation of the penetration resistance specifications. In this stage, the penetration-resistance properties are interpreted in (mapped to) the given system to generate the set of validation-check specifications, which are required for altering/viewing of the global variables and for invoking internal functions. These validation-check specifications are the *penetration-resistance specifications*, since they ensure the penetration resistance for the given system; i.e., they are used to verify that the conditions enabling integrated flows satisfy the penetration-resistance properties.

Stage 3: analysis of integrated execution paths. In this stage, each integrated (execution) flow path (from Stage 1) is analyzed for adherence to the penetration-resistance specifications (from Stage 2) using the model rules, to detect whether the set of required validation-check statements is actually present in the path; if absent, the path is flagged to signify the existence of a possible penetration-related flaw.

3.1 Execution Path Integration

The integrated (execution) flow paths of Stage 1 are derived from the *integration* of the *unit information flows*, the *unit function flows* (flow of control from one function to a second), and the unit *condition statements* encountered while tracing an execution path through the TCB source code. A unit (information or function) flow is caused by a single program statement. For example, an assignment statement $a = b$ causes a unit information flow from b to a; a function call statement *call func2(a)* occurring in function *func1* causes a unit function flow from *func1* to *func2*, as well as an unit information flow from the actual to the formal parameters of function *func2*. A unit condition statement enables a unit (information or function) flow to occur. For example, the conditional expression of an "if" statement is a unit condition statement that enables the information and function flows occurring within the body of the "if" statement.

A unit information flow is represented as a pair of <F:V> elements connected by a arrow. A <F:V> element represents the variable V within function F. A unit information flow given by <F1:V1⇒F2:V2> implies an information flow occurring from variable V1 in function F1 to variable V2 in function F2. (Within a unit flow or condition, the underscore character "_" is used as a wild-card identifier for any individual component that is irrelevant for analysis purposes.) Similarly, a unit function flow <F1↦F2> implies a unit function flow from F1 to F2. A unit condition statement <F:C> represents a unit condition statement C (in "C" language syntax) that occurs within function F.

Within the source code of the TCB of a system, we define the set of all *unit information flows*, the set of all *unit function flows*, and the set of all *unit condition* statements. The *integrated flow path* may then be represented as an ordered set of elements, where each element is a member of the union of the unit information-flow, function-flow, and condition sets. In other words, an integrated execution flow path is a sequential concatenation of the unit flows and conditions encountered along a given execution path within the system source code.

The integrated execution paths that are of interest in Stage 1 above, start at the system interface and end with i) the *altering* of a global variable,

ii) the *viewing* of a global variable, iii) the *invocation* of certain internal functions within the system. For example, an *altering* flow path begins with a unit flow or condition in a system call, SC, and ends with a unit information flow *to* the global variable VAR as shown below:

SC:V1⇒_:V2 *or* **SC:C** *or* **SC↦F** ,
... , **F:V⇒_:VAR**

Similarly, a *viewing* flow path begins with a unit flow or condition in a system call, SC; contains a sequence of unit information flows originating *from* the global variable VAR through a chain of other intermediate variables; and ends with a unit information flow to a variable which is either, i) visible at the user interface (UV), or ii) supplied as an argument (ARG) to an output (e.g., print) function as shown below:

SC:V⇒_:VV *or* **SC:C** *or* **SC↦F**, ... ,
F:VAR⇒F1:V1,
F1:V1⇒F2:V2, ... , **Fn-1:Vn-1⇒Fn:Vn** , ... ,
Fn:Vn⇒ _:UV *or* **F:Vn⇒print:ARG**

Finally, a flow path terminating with the invocation of an internal function will contain a sequence of unit function flows starting at the system call, SC, and ending with the internal function FUNC. These function flows may be interspersed with the unit condition statements that qualify the execution path and the unit information flows from the actual to formal parameters of FUNC (these may be useful to establish the context of invocation of FUNC). A flow path is represented as:

SC:C, ... ,
SC↦F1, **F1↦F2** , ... , **Fn-1↦Fn**,
Fn:V1⇒FUNC:ARG1, ... ,
Fn:Vn⇒FUNC:ARGn,
Fn↦FUNC

One could argue that, for penetration analysis, any condition checks we associate with a function flow could be replaced by conditions associated with information flows from actual to formal parameters or *vice versa*. However, this is not the case. One simple reason is that some functions do not have any formal parameters or return values. Intuitively, there are certain conditions that need to be checked before a process is allowed to invoke certain system functions and this translates naturally to conditions associated with function flows. These conditions are often dependent on the context in which the system function is invoked. The context definition may involve information flows from actual to formal parameters of the function, signifying that the function was invoked with a certain type of parameter.

3.2 Penetration–resistance specifications

Integrated (execution) flow paths begin with system–call interfaces, and include information flows to variables and function flows. This suggests that three types of validations–check specification are necessary, namely (1) interface validation–check specifications, which include parameter validations, or *parchecks*, (2) validation–check specifications for information flows to variables, or *varchecks*, and (3) validation–check specifications for function flows, or *funcchecks*. These validation checks can be *context dependent* or *context independent* (discussed in Section 5.2 in some detail). Context definitions, which must accompany the validation check specification, consist of (1) a condition on functions or variables, (2) one or more function flows, and (3) one or more information flows. In contrast, a context–independent, validation–check specification will not include a context definition.

Interface validation–check specifications: These specifications are derived by interpreting the isolation or tamperproofness properties of a system. The predicates *parchecks* specify the validation checks required at the interface of a system (TCB) call entry.

The syntax of the interface validation–check specifications used in the tool is:
– *context–independent checks:*
parchecks (Entry, [checks], ci, '_'), and
– *context–dependent checks:*
parchecks (Entry, [context], cd, [checks]),
where the *Entry* denotes a system (TCB) entry point, the *context* denotes the context definition, the flag *(ci)cd* denotes the context (in)dependence, *checks* denotes a set of context–(in)dependent, validation–check specifications, and '_' denotes an empty context. For example, the validation–check specification for the **ustat** system call of Secure Xenix:
parchecks (*ustat*, [**buf** *is in user space and is writable*], *ci*, '_'])
is context independent because, regardless of the type of call, parameter, or object, if the call (i.e., ustat) returns a value at an address specified by the user, that address must be in user space and must be writable. In contrast, the validation–check specification for the system call **msgget**:
*parchecks (msgget, [***key** *!= IPC_PRIVATE &* **key** *not found in msq table], cd, [***msgflg** *specifies IPC_CREAT])*
is context dependent because user parameter **msgflg** must be validated to specify the creation of a message queue, when the value of the other parameter **key** is bound in the context–defining condition that the message queue is to be public but does not already exist in the system.

Validation–check specifications for function flows: These specifications determine the conditions under which an internal system function can be invoked by a user–level untrusted process, and are usually derived by interpreting properties of noncircumventability and user/system dependencies. The predicates *funcchecks* specify

the validation checks necessary to invoke such a function.

The syntax of the validation-check specifications for function flows used in the tool is:
- *context-independent checks:*
  funcchecks (Function, ci, [checks], '_'), and
- *context-dependent checks:*
  funcchecks (Function, cd, [context],[checks])

where the *Function* denotes a system (TCB) internal function, the flag *(ci) cd* denotes the context (in)dependence, the *context* denotes the context definition, the *checks* denotes a set of validation-check specifications, and '_' denotes an empty context definition. For example, the validation check specification
funcchecks(panic, ci, ['IMPOSSIBLE'], '_')
is context independent because, regardless of the context of use, the validation check for reaching the system internal function *panic* from the user interface must always fail. In contrast, the validation-check specification
funcchecks(copyseg, cd, [source or destination address is specified by the user], ['IMPOSSIBLE'])
is context dependent because, if the source or destination address for the kernel internal function **copyseg** is specified by a user, then the validation check must always fail.

Validation-check specifications for information flows: These specifications determine the conditions under which the alteration/viewing of a variable is allowed, and are usually derived by interpreting the properties of noncircumventability and consistency of system global variables. The predicates *varchecks* specify the validation checks necessary to either alter or view a global variable through a TCB entry point.

The syntax of the validation-check specifications for information flows used in the tool is:
- *context-independent checks:*
  varchecks (Variable, alter/view, ci, [checks], '_'),
- *context-dependent checks:*
  varchecks (Variable, alter/view, cd, [context], [checks]),

where the *Variable* denotes a system (TCB) global variable, the flag *alter/view* denotes whether the information flow alters or views the variable, the flag *(ci) cd* denotes the context (in)dependence, the *checks* denotes a set of validation-check specifications, and '_' denotes an empty context definition. For example, the specification
varchecks (msgque, alter, ci, [invoking process has write access to msgque], '_')
is context independent because regardless of the context of occurrence, alteration of a message queue (or its components,) always requires write access validation for the current process. In contrast, the validation-check specification
varchecks(proc->p_sig, alter, cd, [PRIV_KILL privilege not present], [current process owns the process being signaled])
is context dependent, because alteration of the process signaling variable *proc->p_sig* requires that the calling process be the owner of the signaled process, in the context that the calling process does not possess the privilege PRIV_KILL.

Note that multiple validation check specifications may be applicable to variable or function flows. Whenever this is true, the *conjunction* of the context-independent validation checks and the *disjunction* of context-dependent validation checks are used for the flow.

3.3 Analysis of the integrated execution paths

The analysis phase is a mechanical process of applying the rules of the penetration analysis model to detect violations of the penetration-resistance properties. The integrated execution paths are analyzed according to the model rules (using pattern-matching techniques) to detect the presence of the required validation checks (derived by interpreting the penetration-resistance specifications.) If a execution path does not include these checks, then that path is deemed flawed and flagged accordingly.

For example, consider the set of validation-check specifications:

varchecks(VAR, alter, cd, [PCtxt1], [F1: C1]).
varchecks(VAR, alter, cd, [PCtxt2], [F1: C2]).
varchecks(VAR, alter, ci, [F2:C3], '_').
varchecks(VAR, alter, ci, [F2:C4], '_').
varchecks(VAR, alter, ci, [F2:C5], '_').

where [PCtxt] denotes a context definition and [Fi:Ci] denotes validation checks represented as condition statements. Using these specifications, we can determine that among the two integrated altering flow paths P1 and P2:

Path P1	Path P2
SC: Cnd1	SC: Cnd1
SC: V ⇒ F1: V1	SC: V ⇒ F1: V1
SC ↦ F1	SC ↦ F1
PCtxt2	PCtxt1
F1: C2	F1: C2
F1: V1 ⇒ F2: V2	F1: V1 ⇒ F2: V2
F1 ↦ F2	F1 ↦ F2
F2: C3	F2: C3
F2: C4	F2: C4
F2: C5	F2: C5
F2: V2 ⇒ F2: VAR	F2: V2 ⇒ F2: VAR

P1 is correct while path P2 is flawed, since after establishing the context by PCtxt1, P2 does not include the validation checks required in that context. Note that, in the above example, only one of the two context-dependent validation checks apply to each flow path, whereas all the context-independent checks apply to each path.

Further analysis, which is outside the scope of our method and tool, is necessary to construct actual scenarios of penetration that take advan-

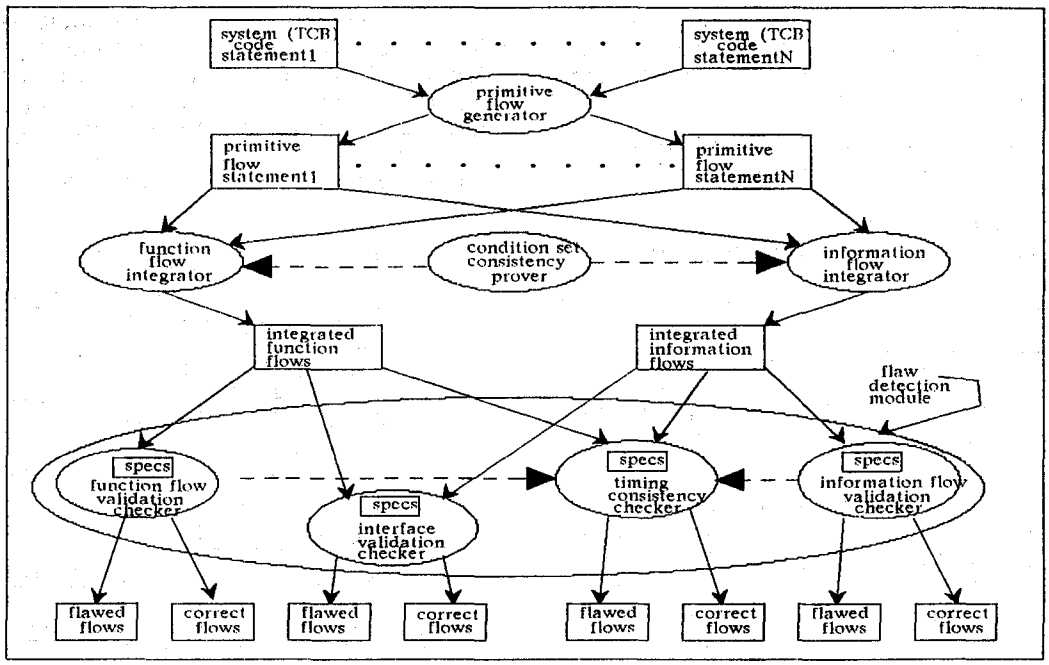


Figure 1. *Automated Penetration Analysis System*

tage of the identified flaws. Actual penetration examples and scenarios are illustrated in detail in Sections 4.2 and 4.3.

3.4 Virtues and Limitations

A major advantage of our penetration-analysis method is that it allows the complete, systematic analysis of a system. It can be used to verify specific penetration-resistance properties of a system's source code, and can be used for automating the tedious and repetitive aspects of penetration analysis. For example, Stages 1 and 3 have been automated (viz., Section 4.1). Stage 2 is performed manually since it does not typically involve repetitive activities. The partial automation of this stage is possible if proper design and programming disciplines are used (viz., discussion in Section 5.2).

Our penetration-analysis method does not address system penetration caused by administrative subversion, by inadequate design or use of the hardware base, by system failures, or by insertion of miscreant code into tools necessary for system generation, distribution, or installation. Instead, it addresses system-penetration patterns caused by unprivileged users' code interactions with a system. Thus, the class of penetration patterns that can be discovered can be characterized precisely using this method.

4. AN EXPERIMENTAL TOOL FOR AUTOMATED PENETRATION ANALYSIS AND ITS APPLICATIONS

4.1 Tool Overview

The development of the Automated Penetration Analysis (APA) tool is based on the penetration analysis method. Figure 1 illustrates the basic structure of the APA tool. The ovals represent system modules while the rectangles represent data that is either input to or output from the modules. Most of the modules (all except the Primitive Flow Generator) have been written in Prolog. We have used Quintus Prolog 2.1 on an IBM RT, model 125, running AIX 2.2.1 for our implementation.

The source code of the system undergoing penetration analysis is the input to the Primitive Flow Generator (PFG). This module converts each C source code statement into one or more Prolog facts called the primitive flow statements. The primitive flow statements record all unit information flows, all unit function flows (call and return statements), all unit condition statements, and sequencing data so that the unit flows can be integrated. The Primitive Flow Generator, which was developed as a part of a earlier covert storage-channel analysis project described in [9,10], is written in C, *lex* and *yacc*. It consists of approximately 4,500 lines of

code, and produces all the primitive flows for the Secure Xenix source code in less than 30 minutes with our experimental setup. Further details may be found in [9, 10].

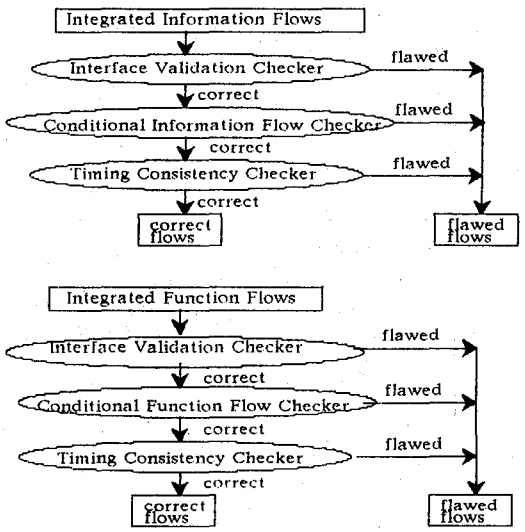
The Information Flow Integrator (IFI) integrates the execution paths between a given entry point (system call or kernel call) at the system interface and a given global variable within the system, while the Function Flow Integrator (FFI) integrates flows between a given system entry point and a given internal function within the system. The flow integrators consist of approximately 3550 lines of Prolog code, and their execution time is exponential in the size of the input program. The structural details and optimization techniques relevant to the integration stage are described in detail in [8].

Information and control flow techniques are used to *integrate* the execution paths within the system source code, and generate integrated flow paths. These paths show, in a sequential manner, all the pertinent information flows, control flows between functions, and the choices made in the conditions encountered along the execution path.

The Condition Set Consistency Prover (CSCP) is used to verify that the unit conditions along a path do not contradict each other. It consists of about 200 lines of Prolog code. The design details can be found in [8].

The Flaw Detection Module (FDM) analyzes the integrated flows or execution paths generated by the flow integrators. It is based on the Hypothesis of Penetration Patterns, and uses the rules of the penetration analysis model to detect penetration flaws in the integrated execution paths. The set of condition statements required for penetration resistance is supplied to the submodules in the form of a database of Prolog specifications derived from an interpretation of the penetration-resistance properties for the system under consideration. The submodules then analyze the integrated flow paths input to the Flaw Detection Module and compare the actual checks present with the set of required validation checks. Whenever a submodule fails to find a specified validation check, it flags that path as containing a penetration related flaw. Thus, it separates the flawed integrated flow paths from the correct ones.

The Flaw Detection Module has four submodules : (i) the Interface Validation Checker – which checks for incorrect/missing validations of entry point parameters and privileges, (ii) the Conditional Information Flow Checker – which detects incorrect/missing validations required for altering/viewing access to global variables, (iii) the Conditional Function Flow Checker – which detects incorrect/missing validation checks required for invoking critical internal functions, and (iv) the Timing Consistency Checker – which looks for timing inconsistencies of condition checks. (Of course, the number of validation-check specifications for each system call depends usually on the call itself; e.g., on the interface conditions and execution paths shared with other calls). The verification of the integrated flow paths is illustrated below.



The user interface of the Automated Penetration Analysis tool consists of a set of commands that either,

a) search for integrated flow paths that lead to a target information or function flow starting from a given set of system entry points, or

b) perform penetration analysis on the set of integrated flow paths found for a given set of entry points and a given target flow, to flag the flow paths that possibly contain penetration-related flaws.

In both cases, the command parameters specify the set of entry points, the target flow and whether one or all such integrated flow paths are to be located.

4.2 An Example of Automated Penetration Analysis

In this section, we present a simple example to illustrate the function of the various stages of the Automated Penetration Analysis tool and to provide some intuition for the practical use of the penetration analysis method.

Figure 2 illustrates the simplified source code of the Secure Xenix system call **ustat** which has two user-supplied parameters *dev* and *buf*. This call returns information about a mounted file system identified by device number *dev*, and writes it out to a location pointed at by parameter *buf*. **ustat** calls the assembly routine **copyseg** to write out to location *buf*.

The kernel internal function **copyseg** is considered a critical function because it copies the contents of one segment to another without checking for read/write access to the segments being read/written and without checking whether the segment selectors refer to user area or system area. The reason of omitting these checks is that, by the time this function is executed, the user's access to objects have been already verified. **copyseg** has three arguments, *src*, *dst* and *cnt* specifying the source and destination segment selectors (for the copy operation) and the number of bytes to be copied.

When the segment of code for **ustat** shown in Figure 2 is input to the Primitive Flow Generator, the output is a set of primitive flow statements. The set of primitive flows are then fed to the Function Flow Integrator (FFI). The FFI integrates (execution) flow paths leading to a given function flow. Figure 3 shows a single integrated flow path including a function flow to **copyseg** starting at the **ustat** system call, first in APA format and then in block diagram format.

The penetration–resistance specifications relevant to this example are shown in Figure 4, first in plain English, then in APA syntax. The Flaw Detection Module uses the specifications to analyze the integrated flow path and arrives at the conclusion that the flow path is flawed since the parameter checks for *buf* cannot be matched in the flow path. Specifically, the flaw appears because **ustat** proceeds to write into the location pointed at by *buf*, but fails to check that *buf* points to a writable location within the invoking user's address space.

The penetration scenario for exploiting this flaw is illustrated in Figure 5. As the figure shows, the user invoking **ustat** can cause the kernel to write

```
SYSTEM CALL
ustat()
{
    register struct mount *mp;
    filsysp_t fp;
    register struct a {
        int dev;
        faddr_t buf;
    } *uap;

    uap = (struct a *) u.u_ap;
    for (mp = mount;
        mp < &mount[v.v_mount];
        mp++) {
        if (mp->m_dev == uap->dev) {
            fp = (filsysp_t)bimap(mp->m_bufp);
            if(copyseg(&fp->s_tfree,uap->buf,
             sizeof(daddr_t)+sizeof(ino_t)) == -1) {
                u.u_error = EFAULT;
            }
            return;
        }
    }
    u.u_error = EINVAL;
}
```

Figure 2. *C–language code of a fictitious* **ustat( )** *system call*

to any memory location (even outside his own address space) and can clobber useful information there. This example illustrates a violation of the isolation property because of inadequate parameter validation and/or absent system/user address space separation checks.

4.3 Experiments Using the Automated Penetration Analysis Tool on Secure Xenix Source Code

Several additional experiments were conducted using the Automated Penetration Analysis tool on the source code Secure Xenix, a Unix® type operating system. A few of them will be described here in detail to illustrate the usefulness of the tool in determining both correct and flawed implementations of the penetration–resistance properties.

In Figures 6 – 11, we show sections of Secure Xenix integrated (execution) flow paths in block–diagram format. For the sake of brevity, we only include selected path components. The following conventions are adopted in illustrating integrated flows: (1) the information flows are represented by rectangles, (2) the function calls and returns are represented by ovals, (3) the conditions checked are represented as diamond shaped boxes, and (4) the sequencing between

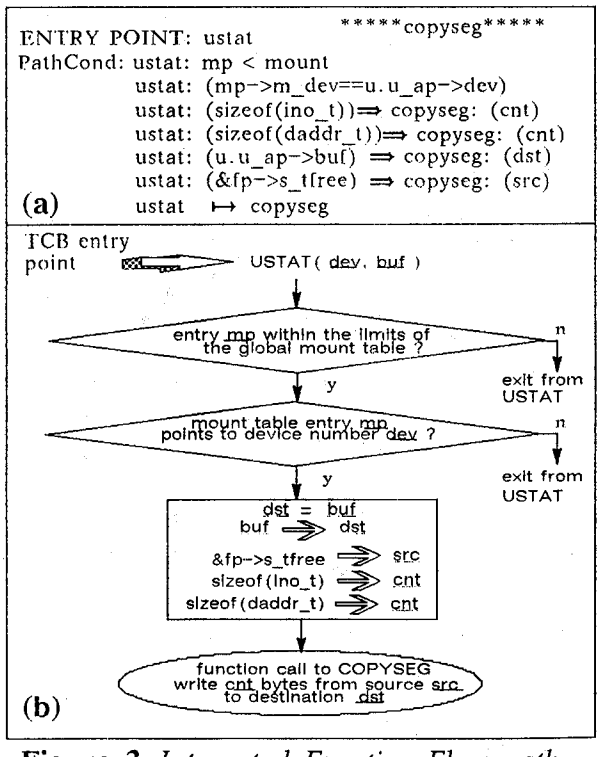


Figure 3. *Integrated Function Flow path to the* **copyseg( )** *internal function* **(a)** *APA format* **(b)** *block diagram*

```
parchecks(ustat,
        [mount table entry corresponding to dev exists]
        ci,
        _).

parchecks(ustat,
        [buf is in user space and is writable],
        ci,
        _).

funcchecks(copyseg,
        cd,
        [source address is supplied by user],
        ['IMPOSSIBLE']).

funcchecks(copyseg,
        cd,
        [destination address is supplied by user],
        ['IMPOSSIBLE']).
```

(a)

```
parchecks(ustat,
        [_,->,m_dev,==,u,'.',u_ap.->,dev],
        ci,
        _).

parchecks(ustat,
        [copyout,'(',_,',',u,'.',u_ap,->,buf,',',_],
        ci,
        _).

funcchecks(copyseg,
        cd,
        [_,[u,'.',u_ap,->,_],_,==>,copyseg,src,0],
        ['IMPOSSIBLE']).

funcchecks(copyseg,
        cd,
        [_,[u,'.',u_ap,->,_],_,==>,copyseg,dst,0],
        ['IMPOSSIBLE']).
```

(b)

Figure 4. *Penetration Resistance Specifications for example (a) English (b) APA syntax*

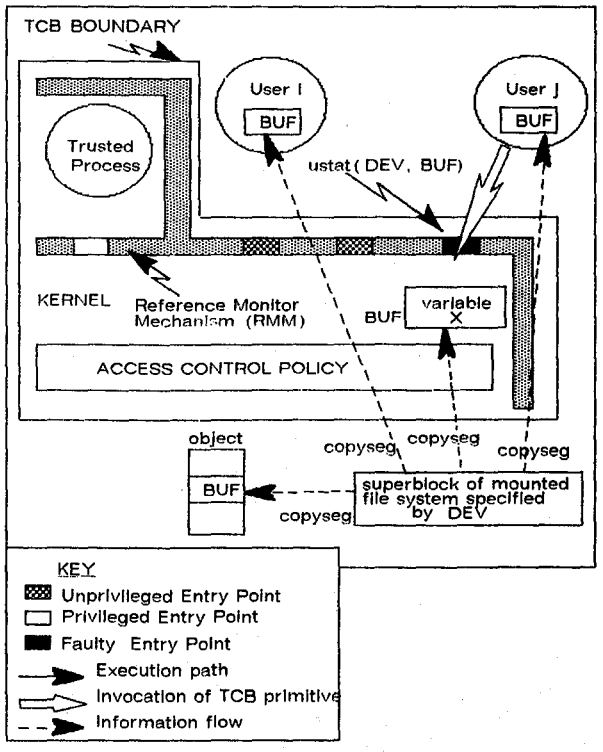


Figure 5. *Penetration Scenario for Fig. 3*

flows, calls, and condition checks are represented by arrows.

4.3.1 Experiments on Kernel Code

Experiment 1. Path to the *panic* function: This experiment illustrates a case of TCB penetration without Reference Monitor Mechanism penetration, which is caused by an undesirable sys-

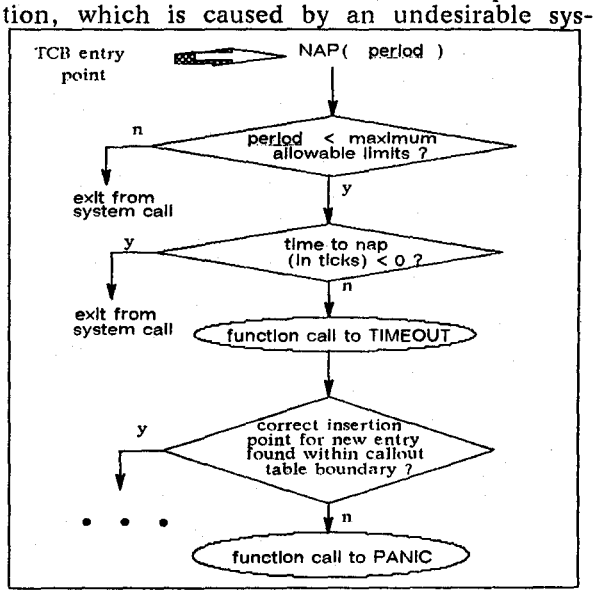


Figure 6(a). *Path to the* **panic( )** *function*

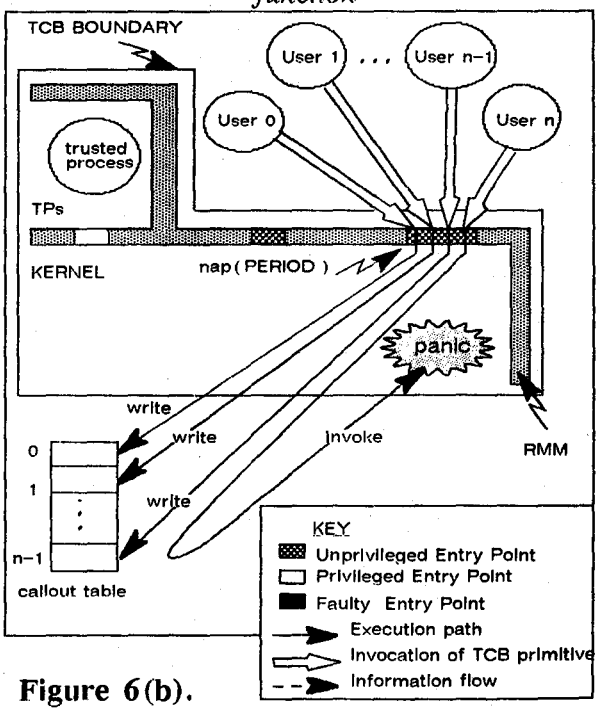


Figure 6(b).

Penetration Scenario for Figure 6(a)

® UNIX is a registered trademark of the Unix Systems Laboratory, Inc.

In Figures 6 – 11, TPs stands for Trusted Processes and RMM for the Reference Monitor Mechanism.

tem/user dependency. We traced the paths from all the system entry points (system calls) to the Secure Xenix internal function *panic*, which causes the system to crash. The experiment revealed that out of the 110 system calls in the system we analyzed, 38 of the system calls could lead an unprivileged user to the *panic* function. In fact, there are 15 independent paths to the *panic* function that could be traced from the user interface.

In Figures 6(a) and (b), we show one such path to the *panic* function and the resulting scenario of penetration. This path starts at the *nap* system call which suspends the calling process from execution for at least the number of milliseconds specified by parameter *period*. The parameter *period* is first checked to be within the maximum allowable range and the number of clock ticks corresponding to the value of *period* is also checked to be greater than zero. Next, the function *timeout* is invoked to insert an appropriate entry into the global *callout* table. The *callout* table is a list of entries each specifying that a certain function is to be called after a certain number of clock ticks. The kernel sorts entries in the *callout* table based on the number of ticks for that entry. The function *timeout* searches the *callout* table for the index of the correct insertion point for the new entry; if this index is beyond the table limits (i.e., the *callout* table would overflow if the new entry is inserted), then the system crashes by invoking the *panic* function. This is unsatisfactory, because a user could deliberately fork a large number of processes each of which invokes system call *nap*, thus causing the system to crash and resulting in denial–of–service to other users.

The scenario in Figure 6(b) typifies an undesirable system/user dependence which should not be allowed to exist within a penetration–resistant system. It must be noted that the *callout* table is a global variable and is not a system defined object. Hence, access to the callout table is not monitored or controlled by the Reference Monitor Mechanism. In fact, there is no evidence of a breach of the Reference Monitor requirements, yet we have a scenario of TCB penetration.

Experiment 2. Paths to the internal function *copyseg*: This experiment illustrates violations of the properties of isolation and noncircumventability through flow paths that lead to invocation of the critical internal function *copyseg*. In this experiment, we traced all the integrated flow paths that lead to the *copyseg* routine starting from the user interface. As mentioned in the last section, in all but two cases, *copyseg* was called to copy from/to segment selectors specified by the kernel. In the other two instances, namely through system calls *ustat* and *shutdn*, *copyseg* was called with user specified segment selectors. One such integrated execution path through the *ustat* system call and the resultant penetration scenario was illustrated in the last section in Figures 3 and 5. Here, we will illustrate (in Figures 7(a) and (b)), a single integrated execution path through the system call *shutdn* and the penetration scenario caused by it.

System call *shutdn* is part of the security operator function, and is used to halt the CPU. Before halting, it updates the information on disk based on the information in core memory and sends out appropriate messages to the console.It has a single argument *addr*. If *addr* is nonzero, it specifies the address of a superblock that is written to the root device before the CPU is halted. This feature facilitates filesystem repair when the root superblock is corrupted and has to be replaced. *Shutdn* can be called only by a user possessing the privilege SHUTDN, which is possessed by the security operator role.

As seen in Figure 7(a), the SHUTDN privilege is first checked and then a function flow to internal function *shutdown* occurs. Next, the user parameter *addr* is checked to be nonzero before the function *copyseg* is called to write from the address specified by *addr* to the root superblock.

A problem of role separation arises here, because *addr* is an address specified by the user, and the execution path reveals that *addr* is not checked (to point to a readable location within the user's address space) before a *copyseg* operation writes the contents to the root superblock. Thus, as shown in Figure 7(b), *addr* can point to any system area (including objects belonging to other system administrators) and the contents of *addr* are written to the root superblock which happens to be universally readable. This is a direct breach of role separation [13], and represents a TCB penetration scenario which is caused by inadequate implementation of the Reference Monitor Mechanism.

Experiment 3. Alteration of the global message queue table : This experiment reveals a violation of the property of timing consistency of condition checks. In system call *msgsnd*, the operations of access authorization and actual access are not done in an atomic sequence resulting in a potential timing inconsistency.

In a Unix–type system, a message queue (msgque) is a mailbox where the order of message arrival is maintained and senders and receivers are processes. The global msgque table has entries that describe all the message queues currently in existence in the system.

In Figures 8(a) and (b), we illustrate an integrated flow path originating at the *msgsnd* system call and leading to an alteration of the message queue table, and the resulting scenario of penetration. This system call is used to send a message to a queue identified by parameter *msqid*. Parameter *msgp* points to a data structure con-

taining the message type and the message text; *msgsz* specifies the size of the message text in bytes; *msgflg* specifies the action to be taken when certain error conditions occur.

The system checks that *msqid* points to a valid message queue and that the calling process has write access to that message queue. Memory location *msgp* (which has to be read from) is then validated to lie within user space readable by the

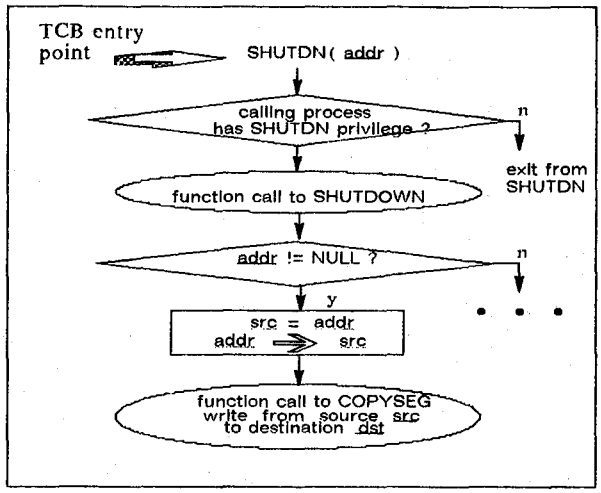


Figure 7(a). *Integrated Function Flow path to the* **copyseg( )** *internal function*

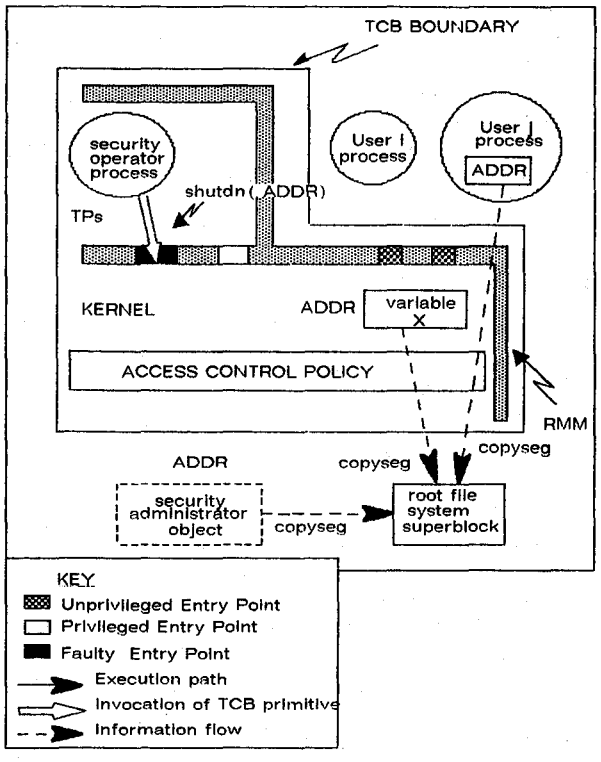


Figure 7(b). *Penetration Scenario for Figure 7(a)*

calling process. The system then checks for availability of the three types of resources necessary to add the new message to the message queue, namely, i) there is enough space on the message queue to add the new message, ii) there is a free message header available, and iii) there is enough physical space in the message map to hold the entire message text. If any one of these resources is not currently available, the system goes to sleep waiting for the resource to become available and once it returns from sleep the system checks to see if the message queue still exists

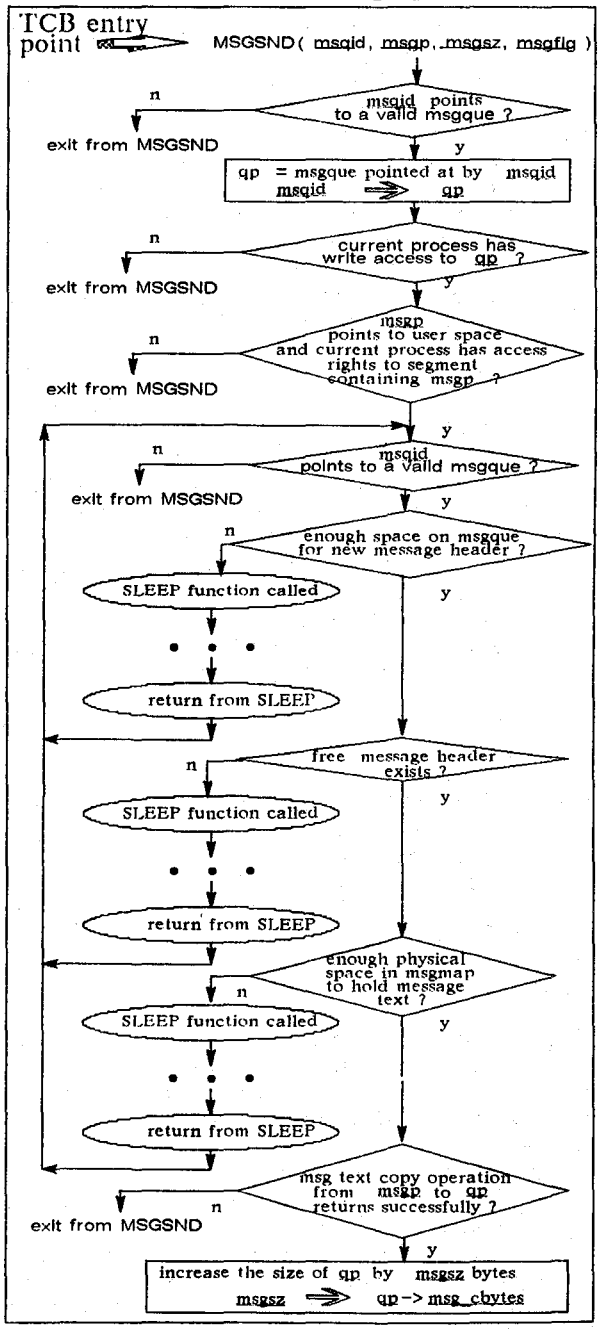


Figure 8(a). *Flow path with timing error*

before going on. In fact, only when all the resources become available and the message queue is checked to still be in existence, will the message actually be put on the queue. At that time, the size field of the message queue (global variable *msgque –> msg_cbytes*) will be altered along with several other fields.

The validation check for the write access to the message queue is performed early in the execution path. This check is required, however, for the altering flow to the global variable *msgque –> msg_cbytes*. Since the process can go to sleep several times between the access check and the actual information flow, the effect of this validation check is not assured at the time the flow occurs. Thus, this integrated flow path violates the property of timing consistency of validation checks. It is possible that at the time of message queue alteration, the access rights of the message queue have been changed (by another user process invoking the *msgctl* system call) while the calling process slept, and the calling process no longer has write access to it.

This experiment shows a scenario where the timing consistency of a validation check is not properly enforced by the Reference Monitor Mechanism, leading to a scenario of Reference Monitor as well as TCB penetration.

4.3.2 Experiments on Trusted Process code

Experiment 4. Timing Inconsistency in trusted process mkdir: In this experiment, we illustrate a scenario where a potential timing inconsistency exists within a trusted process. In Unix–based systems, the kernel maintains consistency mainly through a discipline of non–preemption. Trusted processes, however, do not enjoy the luxury of non–preemption and hence need other methods such as locks and ignoring signals to maintain timing consistency of the condition checks.

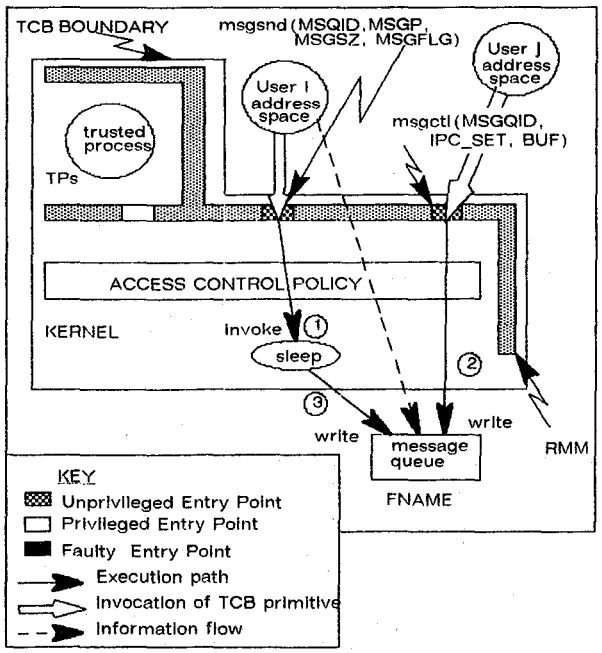


Figure 8(b). *Penetration Scenario for Figure 8(a)*

In Figures 9(a) and (b), we illustrate the integrated flow path that causes timing inconsistency in trusted process *mkdir* and the penetration scenario that results from it. The trusted process *mkdir* is used to create directories in the filesystem hierarchy. One or more directories may be created depending on the number of arguments supplied. There is a *–s* (security) option which allows the invoker to create directories with a specified higher security level, not lower than the level of the parent directory and not higher than the file system maximum security level. System call *setflbl* allows a privileged user to set the level of a file to any arbitrary value.

In this integrated flow path, the trusted process *mkdir* is invoked with one directory argument *fname* and the –s option with option argument *level*. That is, *mkdir* is requested to create a directory named *fname* with security level *level*. As shown in the Figure 9(a), the length of the string given by argument *fname* is checked to be shorter than the maximum allowable length for a directory. Then, write access to the parent directory (of *fname*) is checked for the calling process. Next, if the –s flag is specified, and if the calling process does not possess the MAC_EXEMPT privilege, security level *level* is validated to be equal to the security level of the parent and no higher than the filesystem maximum level. These compatibility checks ensure that the security level hierarchy in the filesystem (where, the security level of a parent is no higher than the child, and the filesystem maximum level is no lower than the level of the files within it) is maintained. Then after a number of other operations – that actually create the new directory *fname* and initialize it – system call *setflbl* is invoked to set the security level of the new directory to *level*.

Since, trusted processes are preemptible, it is obvious that in the integrated execution path shown in Figure 9(a), the *mkdir* process can be preempted between the security level compatibility check and the call to *setflbl*. It is quite possible that another process takes control of the CPU within this time interval and raises the security level of the parent directory so as to make it higher than *level*. However, when *mkdir* regains the processor, it calls *setflbl* in privileged mode and sets the level of *fname* to *level*, which is now higher than the level of the parent directory, thus directly violating the access–control policy of the system. To maintain timing consistency, *mkdir* should have placed a write–lock on *parent* while performing the compatibility check and then calling *setflbl* to set the label of *fname*.

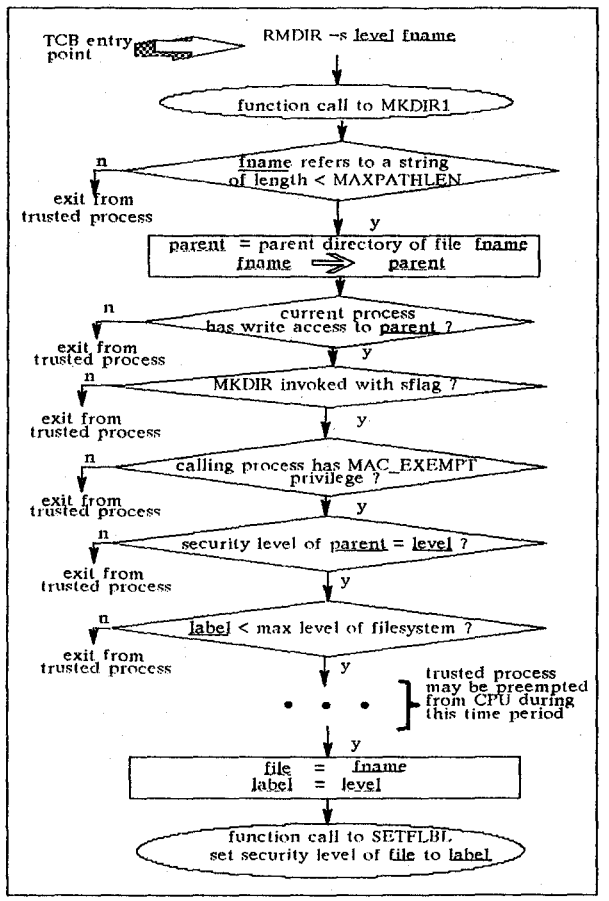


Figure 9(a). *Integrated Flow Path to the privileged system call* **setflbl( )**

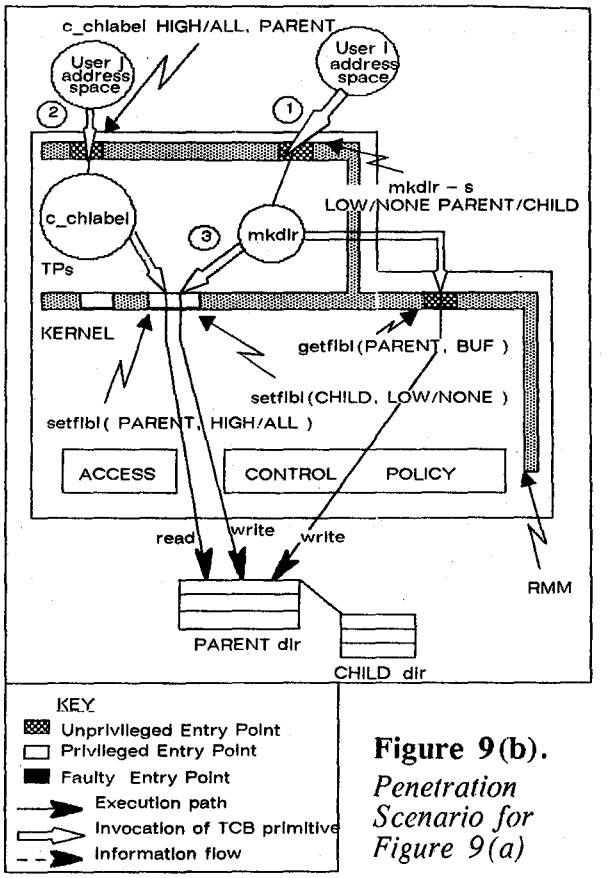


Figure 9(b). *Penetration Scenario for Figure 9(a)*

The timing inconsistencies of this experiment causes a violation of access control policy of the TCB without violating the requirements of the Reference Monitor Mechanism. This is the case because the timing inconsistency occurs in a trusted processes access which, in fact, legitimately circumvents the Reference Monitor validation checks.

Experiment 5. Timing Inconsistency in Trusted Process rmdir: This experiment reveals another timing inconsistency in a trusted process that leads to a penetration flaw. Here we have a TCB penetration without a penetration of the RMM, for the same reason as that with Experiment 4; i.e., the trusted process causing the timing inconsistency legitimately circumvents the Reference Monitor validation checks.

In Figures 10(a) and (b), we illustrate a flawed function-flow path within the trusted process *rmdir*. The trusted process *rmdir* is used to remove one or more directories specified as user parameters. In the figure, *rmdir* is given a single parameter *fname* (or *parent/child*) specifying the directory that is to be removed. The trusted process checks whether the supplied filename exists and is a directory not identical to the current directory. It then checks for read access to *fname*, and reads the file to check if it is a empty directory. If so, write access to the parent directory of *fname, parent,* is checked. If present, *fname* is removed from the filesystem by unlinking it from its *parent* .

The integrated execution path shown in the Figure 10(a) leads from the *rmdir* trusted process interface to a function call (or function flow) to the *unlink* system call. *unlink* writes into a directory and, hence, it checks the write access for all non-privileged processes invoking it. However, *rmdir* is a privileged process with the MAC_EXEMPT and DAC_EXEMPT privileges, hence, the access check modules inside the kernel are bypassed by *unlink* in this situation. Thus, when *unlink* is invoked in a privileged mode, it requires that a write access check be performed in an atomic sequence with the invocation of *unlink* to maintain timing consistency of the access check. Since, trusted processes are preemptible, it is obvious that in the integrated execution path shown in Figure 10(a), the *rmdir* process can be preempted between the access check and the call to *unlink*. It is quite possible that another process takes control of the CPU within this time interval and changes the access permissions on *parent*. However, when *rmdir* regains the processor, it calls *unlink* in privileged mode and writes into a directory to which it may no longer

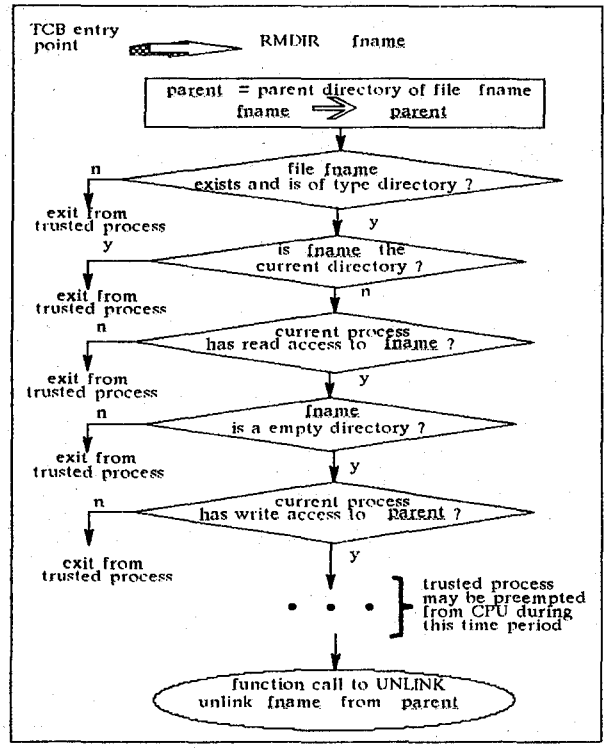


Figure 10(a). *Integrated Flow Path to the system call* **unlink** *in privileged mode*

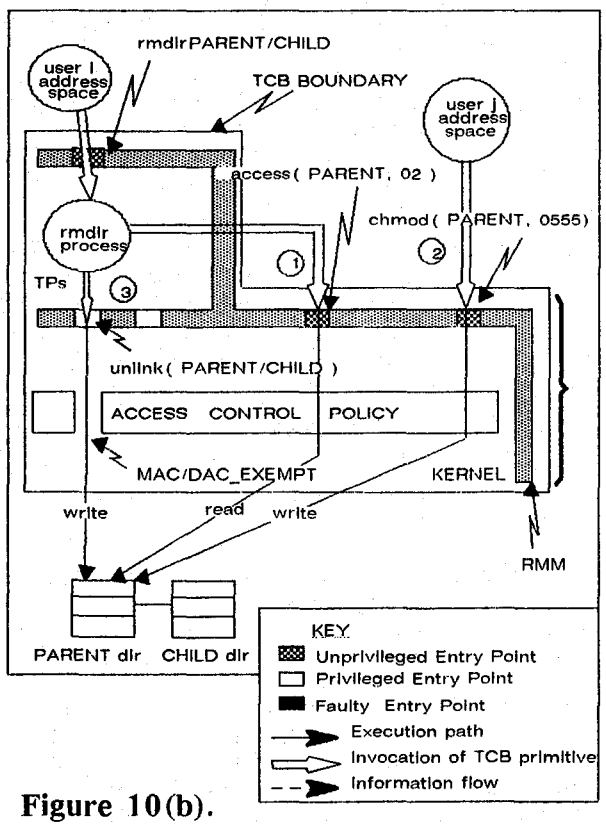


Figure 10(b).
Penetration Scenario for Figure 10(a)

have write access. To maintain timing consistency, *rmdir* should have placed a write–lock on *parent* while checking for write access and then calling *unlink* to write into *parent*.

Experiment 6. User Parameter Validation Error in Trusted Process rmdir: This experiment revealed a penetration scenario caused by inadequate parameter checking at a trusted process interface. In the Figures 11(a) and (b), *rmdir* is also given a single parameter *fname* which points to a string that specifies the name of the directory that is to be removed. The first action of the trusted process is to copy the string referenced by *fname* into a fixed length local buffer *buf* allocated on the user stack. This is done by invoking the subroutine *strcpy*, which copies one string to another without checking if the length of the second string is large enough to accommodate the first string. The subroutine *strcpy* presupposes that adequate length checking of the arguments to *strcpy* was done.

This experiment illustrates a case of inadequate parameter validation by a trusted process, since

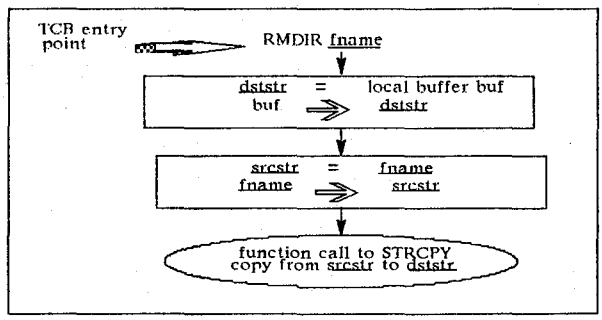


Figure 11(a). *Integrated Flow Path to the subroutine* **strcpy( )**

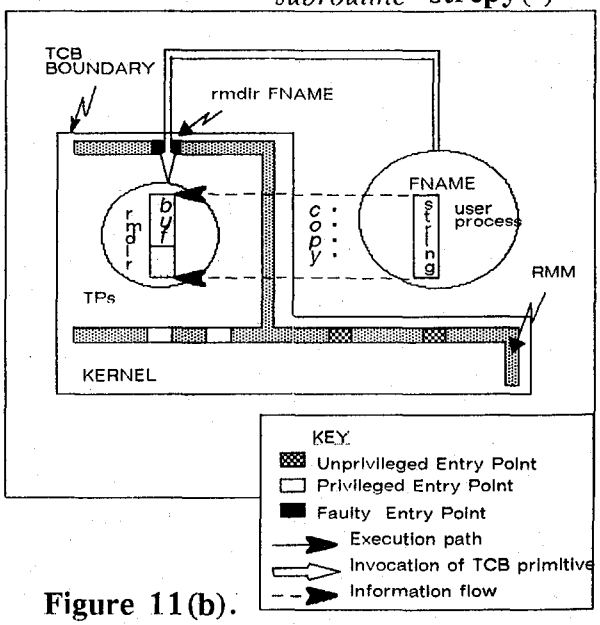


Figure 11(b).
Penetration Scenario for Figure 11(a)

a user specified parameter was not checked to be within the range of length allowed by the system, before being copied into a fixed length buffer on the process stack. The result is that *buf* may be overflowed and a false frame is created on the trusted process stack. This false frame can specify any arbitrary address as the return address of a function call. Since *rmdir* has special privileges, the penetrator can cause any code to be executed with the special privileges of *rmdir*.

Note that the validation of the string length could not have been performed by the Reference Monitor code since that validation if trusted–process–call dependent; i.e., only the trusted process knows the maximum size of the buffer, *buf*, meant to receive the user–specified string. Thus, only the TCB, and not the Reference Monitor Mechanism could eliminate this penetration flaw.

5. USING THE PENETRATION ANALYSIS METHOD – GENERAL OBSERVATIONS

5.1. Scope of Penetration Analysis

The boundary of a system that is subject to penetration analysis can be viewed as the boundary of a Trusted Computing Base (TCB) defined to contain the "security relevant portions of a system" [12]. Thus, in addition to the access–control policy modules, which are encapsulated by the Reference Monitor Mechanism, the TCB usually contains other security relevant modules, such as those for audit, identification and authentication, and trusted path.

The concept of a *reference monitor* was introduced in the Anderson report [1] to be an element of a secure system "which enforces the authorized access relationships between the subjects and objects of the system." The Reference Monitor Mechanism (RMM) is required to be (i) tamperproof, (ii) always invoked, and (iii) small enough to be verifiable [1,12]. Since the RMM only refers to the access control policy, and its verifiability is only with respect to the invariants of that policy, the RMM can be viewed as being a strict subset of the TCB. The RMM tamperprooofness and noncircumventability provide assurance regarding the integrity of the RMM and, implicitly, of the RMM implementation of the access–control policy.

It has been commonly believed that if the RMM tamperproofness and noncircumventability requirements are supported in a TCB, then the penetration–resistance of the entire TCB can be asserted. However, the experiments presented in Section 4 show that penetration resistance of a TCB cannot be guaranteed by these RMM requirements alone, even if we assume that other system policies and mechanisms (e.g., identification and authentication, audit) are penetration resistant(*). This is the case for, at least, the following three reasons:

– TCB penetration may be caused (1) by references which alter internal variables that are not part of any object, or (2) by users' invocations of the TCB that cause a critical internal TCB function to be invoked (viz., Experiments 1 and 2). Therefore, the RMM cannot mediate these accesses since, by definition, it can only mediate subjects' access to the defined objects;

– TCB penetration may be caused by flawed implementation of the penetration–resistance properties in trusted processes that may, otherwise, legitimately bypass the RMM (viz., Experiments 4 and 5). In such cases, the RMM is, by definition, unable to mediate these accesses;

– TCB penetration may be caused by inadequate context–dependent checking of parameters by trusted processes (viz., Experiment 6). In such cases, the RMM is unable to enforce TCB isolation properties, by definition, since it cannot be expected to understand context dependencies of trusted processes.

Experience with our penetration analysis method shows that the set of penetration–resistance properties for a TCB are a strict superset of the RMM requirements, and these properties have to be observed to ensure the security of TCB and entire system. Furthermore, the tamperproofness and noncircumventability properties of a system (e.g., TCB) are a strict superset of the RMM requirements, in the sense that they apply to the whole TCB as opposed to only the RMM. Thus, a penetration scenario caused by a violation of (1) consistency of global system variables, (2) timing consistency of validation checks, and (3) undesirable system/user dependencies, may illustrate TCB penetration without RMM penetration.

5.2 Validation–Check Dependencies on Design and Programming Disciplines

The validation checks that should be included in a system's source code (and specified for use by the Flaw Detection Module) depend on both the design and programming disciplines used in system development. Design–level validation dependencies are common to most operating systems, whereas programming–level dependencies are specific to the actual disciplines used. Identifying both of these types of dependencies is important for the correct (automated or manual) generation of validation–check specifications for use by the Flaw Detection Module. Without the benefit of a correct set of validation–check specifications, the penetration analysis process loses precision.

(*) It should be obvious that one could penetrate the identification and authentication mechanism without penetrating the RMM.; e.g., via password attacks.

Validation checks for penetration resistance are of two major types: interface validations involving parameters and privileges at the TCB interface, and functional (correctness) validations required elsewhere within the system. Typically, the isolation property dictates the validations required at the interface, whereas the other properties determine the validations required elsewhere.

5.2.1 Examples of Design-level Dependencies

The penetration analysis experiments reveal several types of design-level dependencies of validation-checks. Combinations of such dependencies may arise in specifying validation checks for individual system calls.

Call-Dependencies of Validation Checks: Validation checks (or condition check statements) required by penetration-resistance are sometimes dependent on the particular system call in which they occur. For example, in Unix, the global variable *inode* may be altered via both the *mknod* and *write* system calls. Due to the difference in the semantics of the two system calls, however, *write* alters an *inode* that already exists, whereas *mknod* alters an *inode* it has just created on behalf of the user invoking the system call. Thus, the validation-check specification for the altering flow through *write* must state that a validation of the write access to the inode is required (based on some policy (*)) . In contrast, the validation-check specification for the altering flow through *mknod* will not include this write access check. Validation checks that depend on the system call in which they occur are named "call dependent." It should be obvious that *all* interface validations are call dependent, since the very definition of the parameters and privileges are implicit within the system call in which they occur. However, functional validations may or may not be call dependent.

Type-Dependencies Validation Checks: Validation checks may also be dependent on the type of object the system (e.g., kernel) is working on or on the type of command/argument specified at the user interface. For example, within the Secure Xenix system call *semctl*, user parameter *arg* specifies a pointer to a buffer to be either read or written depending on the value of the parameter *cmd*. If *cmd* is equal to IPC_SET, the system call reads from the buffer, and hence a validation check for reading from the buffer is required (based on some policy (*)); whereas if *cmd* is equal to IPC_STAT, the system call writes to the buffer, and hence a validation check for writing to the buffer is required (based on some policy (*)). Thus, depending on the type of the parameter *cmd*, the parameter *arg* is

* Note that the policy under which the access is deemed valid is irrelevant to whether the validation is performed.

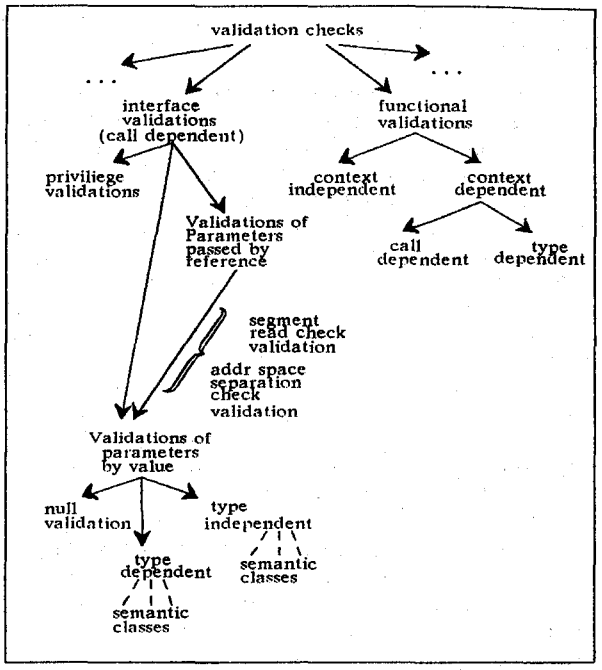


Figure 12. *Examples of Validation-Check Dependencies*

validated in different ways and, thus, different validation specifications must be provided. Similarly, in the system call *aclquery* of Secure Xenix, which queries the access control list of an object, the validation checks are different for different object types specified via the user interface. This kind of dependence of a validation check is termed "type dependence."

Since "call dependence" and "type dependence" are both related to the context in which the user is requesting service from the kernel, we will group them together as *context dependence*. In general, validation-check specifications may be either "context dependent" or "context independent" (see Figure 12). The "context" of those specifications is provided by an execution path, which is determined by the kernel entry point and the entry point parameters.

Dependencies of Interface Validations: The validations required at the user interface include, (i) parameter validations, (ii) privilege validations and (iii) user/kernel address space separation checks. System call parameters can be passed either by value or by reference. The latter type of parameters have to be read into system space to convert them to parameters by value. During the reading in of these parameters, the system should check for (segment) read access as well as whether it is reading from the address space of the invoker, as illustrated in Figure 12. Parameters passed by value can be further classified as either "type independent" or "type dependent" (e.g., parameter *arg* in *semctl*), or as requiring no validation whatsoever (e.g.,

when a parameter specifies a numerical value that has no legality bounds associated with it).

Semantic Dependencies of Parameter Classes: During the analysis of the parameter validation for the Secure Xenix kernel, we have found that it is possible to classify the system call parameters based on call semantics, and that, in the majority of cases, parameters belonging to the same *semantic class* are validated in an almost identical way. Such classification simplifies the process of generation of the set of parameter validations that need to be performed at the user interface. For example, in Secure Xenix, there are a large number of system call parameters that specify a *filename* in the form of a character string. Every single such *filename* parameter is validated in exactly the same way (through the internal function *namei*) to ensure that the string indeed represents a valid *filename* in the system and that the calling process has search access to all the directories in the pathname. Similarly there are other obvious semantic classes such as read and write *buffers*, file *descriptors*, message or semaphore *identifiers*.

Theoretically, every single system call parameter can be placed in a semantic class and the interface validations required for each such class can be clearly specified. When a complete set of semantic classes is obtained, the automated analysis of system call parameter validation becomes possible for a given system by making the specifications of the required validation checks available to the Automated Penetration Analysis tool.

Semantic classes with a large number of members are especially suitable for automated analysis. However, some of the semantic classes will have a single member, and hence, in those cases, automated analysis will involve as much effort as manual analysis of the source code. System call parameters that are *type independent* can usually be placed in large-sized semantic classes. In contrast, type dependent parameters are the ones that are the most difficult to classify and usually belong to single member semantic classes.

5.2.2 Examples of Programming Discipline Dependencies for Trusted Processes

Explicit Object Sharing Among Trusted Processes: Trusted processes of Secure Xenix, and those of most Unix systems, rarely interact with each other except inside the kernel via system calls. Shared global variables between trusted processes exist in a few cases but, in general, trusted processes share no global variables, except in the form of shared (system) objects such as files, pipes, and these objects are shared through system calls. Since the kernel is analyzed for penetration-resistance separately, trusted processes that share no global program variables can be analyzed as separate entities. However, trusted processes that do share global program variables amongst themselves, (such as the "lp" subsystem commands,) must be analyzed collectively as a subsystem. Their relationships with respect to the shared objects must be explicitly defined such that the required validation checks can be specified.

Explicit Parameter Passing to Trusted Processes.
Parameter Passing: Unlike the user specified parameters of kernel calls, the user-specified parameters of trusted processes are not as easily identified syntactically in the code. This is because a single variable might specify a string of parameters passed by reference. When these parameters are validated, the validation condition statements refer repeatedly to the same string identifier (the pointer to the string identifier is advanced successively to refer to the successive parameters referred by the string). Thus, it becomes very difficult to uniquely identify the individual user parameters, and hence to derive the syntactic form of the penetration-resistance specifications, whenever parameter passing is not explicit.
Interactive Input: Many user parameters to trusted processes are input interactively from the user interface and are copied into local variables of trusted processes before validation. The identification of which local variables receive user-specified values via interactive input should be explicit. Otherwise, deriving the penetration-resistance specifications becomes as difficult as analyzing the source code manually and is hence not cost effective.

Explicit Specification of Trusted Process Protection Mechanisms: The majority of the protection mechanisms used by the trusted processes depend on properties possessed by the executable modules of trusted processes and are not always apparent by an analysis of the code. Such mechanisms include setuid/setgid mechanisms, special users and groups (representing administrative roles) with discretionary access to special administrative files and data structures, and special privileges which are endowed directly (and not explicitly acquired). The use of such mechanisms in trusted processes should be made explicit. Otherwise, a potentially large number of extra facts must be manually fed into the Flaw Detection Module thereby reducing the degree of automation possible.

Whenever trusted processes are relatively small (in terms of lines of source code) it may be easier to perform the flaw detection manually by assuming that the trusted process is endowed with power which is not obvious through code inspection. In such cases, automated analysis of trusted processes may not always be cost effective. In these cases, trusted processes can be

analyzed manually using the model and the penetration-resistance properties as guidelines.

5.3 Interpreting the Penetration-Resistance Properties

To generate penetration-resistance (i.e., validation-check) specifications at the source-code level of a system, we must interpret the abstract penetration-resistance properties discussed in Section 2 above in source code. In conducting the experiments presented in Section 4.3, we found that, instead of trying to interpret the abstract properties in the source code directly in a single step, it is easier to first interpret these properties using internal design-level specifications. This additional step enabled us to generate a set of design-level (concrete) properties from the abstract properties, through a study of the system documentation, and then to interpret these concrete properties in the source code. These design-level, penetration-resistance properties can then be used to derive the set of validation-check specifications for verifying that the actual source-code level checks for *alter/view/invoke* accesses are correctly performed within the system.

The process of interpreting abstract penetration-resistance properties in design-level specifications and then in source code is analogous to that of performing model interpretation in a systems' descriptive/formal top-level specifications (DTLS/FTLS) and source-to-code correspondence. However, the DTLS/FTLS differ from the design specifications needed for interpreting penetration resistance properties. In general, the DTLS/FTLS are intended to define the system behavior in terms of user-level objects (e.g., processes, files, directories) and, therefore, do not include specifications of internal system behavior. Thus, DTLS/FTLS are usually not detailed enough to reveal the flaws that cause system penetrability. For example, no amount of analysis at the DTLS/FTLS level documentation would have revealed the timing consistency flaw illustrated in Experiment 3, or the use of *copyseg* to copy to/from a user supplied address illustrated in Experiment 2. Instead of DTLS/FTLS, internal system specifications should be used to derive the concrete properties of penetration-resistance from the abstract properties.

The penetration analysis of a given system can be done in either of two ways. We can start from the abstract properties to generate the concrete properties, which in turn may be used to generate the set of required source code level checks. This is the approach we have adopted in our Automated Penetration Analysis tool implementation, where our method ensures that the required checks are indeed present in the integrated flow paths. Alternately, the set of integrated flow paths could be used to determine whether validation checks in source code satisfy the abstract properties. This approach would be advisable when we attempt to formally verify the penetration-resistance properties of a kernel (see Figure 13).

6. DISCUSSION

The experimental Automated Penetration Analysis tool based on the penetration-analysis method proposed in this paper has been used in several experiments on the Secure Xenix source code, a few of which have been reported here. The tool may be used to detect violations of additional penetration resistance properties. It may also be used for other Unix systems implementing the same set of properties. Furthermore, by merging new flows with old flows, and new checks with old checks of the same system (e.g., TCB), the tool can be used for incremental analysis (of penetration resistance) of updates. Lastly, it can be used for penetration analysis in other applications; e.g., database management systems.

Our observations regarding the separability of the policy concerns from those of penetration resistance, and the insufficiency of the reference monitor mechanism in providing assurance regarding penetration resistance of a system, helps delimit the usefulness of the Reference Monitor properties in penetration analysis. Designs of secure systems can also benefit from the observation that the design and programming disciplines of a system have a large impact on the ease (or difficulty) of performing (automated) penetration analysis. Finally, the observation that the assurance process for penetration resistance is similar to that for policy implementations will allow the use of well-accepted assurance techniques for the purpose of penetration analysis.

We believe that our research is a first step in systematic penetration analysis. However, there remains much work to be done in this area. More research is required to enrich and augment the set of penetration-resistance properties documented in this paper and in [7]. Further work can also be targeted towards developing techniques for (partially) automating the derivation of penetration-resistance specifications for a

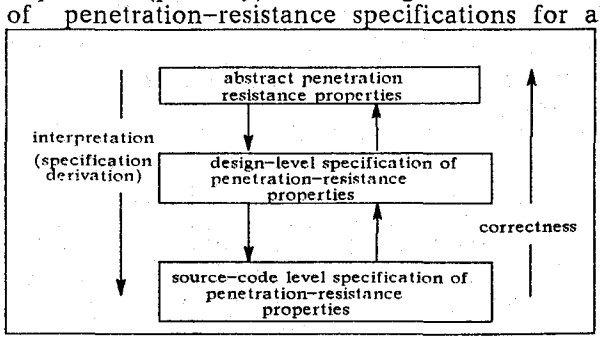


Figure 13. *Derivation of Penetration Resistance Specifications*

given type of system. This suggests the need for a more intuitive syntax for the "context" and "check" portions of these specifications. The current version of the Automated Penetration Analysis tool can also benefit from further optimization to improve its execution speed, and the development of new Primitive Flow Generators for other languages for analysis of system source code written in languages other than C.

ACKNOWLEDGMENTS

Experiment 1 was suggested by the observation that some scenarios of exploiting covert storage channels may, in fact, lead to penetration [15]. Experiment 5 was suggested by Robert H. Morris. Experiment 6 was inspired by Robert T. Morris' experiment with the internet *fingerd* [4].

This work was supported by the IBM Corporation, Federal Systems Company, in Gaithersburg, Maryland under contract number 319429 at the University of Maryland. We are grateful to Tom Tamburo, Wen-Der Jiang, Marty Simmons, Curt Symes, and Tom Russell for their support and encouragement.

REFERENCES

[1] Anderson, J. P., "Computer Security Technology Planning Study, Volume2," *NTIS: AD-772 806*, NTIS, October, 1972.

[2] Bach, Maurice J., *The Design of the UNIX Operating System*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1986.

[3] Bisbey, R., G. Popek and J. Carlstedt, "Protection Errors in Operating Systems: Inconsistencies of a Single Data Value Over Time," USC / Information Sciences Institute, ISI / SR-75-4, December 1975.

[4] Eichin, M.W., and J.A. Rochlis, "With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988," Proc. of the 1989 IEEE Symposium on Research in Security and Privacy, Oakland, California, May 1989, pp. 326-344.

[5] Gligor, V. D., "A Note on the Denial-of-Service Problem," Proc. of the IEEE Symposium on Security and Privacy, Oakland, CA, April 1983, pp. 139-49 (also in *IEEE Transactions on Software Engineering*, SE-10, No. 3, May 1984).

[6] Gligor, V.D., *et al.*, "Design and Implementation of Secure Xenix," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2, February 1987, pp. 208-221.

[7] Gupta, S. and V. D. Gligor, "Towards a Theory of Penetration-Resistant Systems and its Applications," Proc. of the 4th IEEE Workshop on Computer Security Foundations, Franconia, NH, June 1991, pp. 62-78 (to appear in the *Journal of Computer Security*, 1992).

[8] Gupta, S. and V. D. Gligor, " Experience with a Penetration Analysis Method and Tool," Computer Science Technical Report No. 2881, University of Maryland, College Park, April, 1992.

[9] He, Jingsha and V. D. Gligor, "Information Flow Analysis for Covert-Channel Identification in Multilevel Secure Operating Systems," Proc. of the 3rd IEEE Workshop on Computer Security Foundations, Franconia, NH, June 1990, pp. 139-48.

[10] He, Jingsha, *An Automated System for Covert-Channel Analysis in Multilevel Secure Operating Systems*," Ph.D. Dissertation, Department of Electrical Engineering, University of Maryland, August 1990.

[11] Linde, R. R., "Operating Systems Penetration," Proceedings of the National Computer Conference, vol. 44, AFIPS Press, Montvale, N.J. 1975.

[12] National Computer Security Center, *Trusted Computer System Evaluation Criteria*, DoD STD-5200.28, December 1985.

[13] National Computer Security Center, *Trusted Facility Management Guideline*, NCSC-TG-015, Version 1, 18 October 1989.

[14] National Computer Security Center, *Security Testing Guideline*, NCSC-TG-023, (Draft), October 1989, (viz., Section on specification-to-code correspondence).

[15] Tsai, C.-R., V.D. Gligor, and C.S. Chandersekaran, "A Formal Method for the Identification of Covert Storage Channels in Source Code," Proc. of the IEEE Symposium on Security and Privacy, Oakland, California, April 1987, pp. 74-86 (also in *IEEE Transactions on Software Engineering*, Vol. SE-16, No. 6, June 1990, pp. 569 - 580).

EXTENDING OUR HARDWARE BASE: A WORKED EXAMPLE

Noelle McAuliffe

Trusted Information Systems, Inc.
3060 Washington Road
Glenwood, MD 21738

Abstract

In January 1991 Trusted XENIX[1] received a B2 TCSEC rating from the National Computer Security Center (NCSC) [1]. The scope of this evaluation included the following hardware bases: IBM[2] PC AT, IBM PS/2 Models 50, 60, 70, 70P, 70T and 80. This paper describes how Trusted Information Systems, Inc (TIS) extended their evaluated hardware base to include additional platforms produced by different vendors successfully demonstrating that they had maintained Trusted XENIX's B2 rating. Although Rating Maintenance (RAMP[3]) has not yet been endorsed for B2 and above products, our experience provides evidence that it is a viable approach for addressing changes even to higher level products without incurring the cost of performing full evaluations.

Introduction

In June 1989 TIS obtained the rights to Secure XENIX from IBM while it's evaluation was underway. TIS continued development and trust analysis on their new product, Trusted XENIX, and negotiated with NCSC to continue the B2 TCSEC evaluation of the system. The scope of this evaluation included the following hardware bases: IBM PC AT, IBM PS/2 Models 50, 60, 70, 70P, 70T and 80. While completing the initial evaluation, we approached our evaluation team about expanding the list of evaluated hardware platforms. A proposal was made to the team explaining how we planned to demonstrate that the new hardware bases were compatible. The unprecedented nature of this request raised many questions whose considerations would have resulted in an unacceptable schedule delay of the evaluation. TIS decided to proceed with the evaluation as scheduled and to revisit the clone issue after obtaining the initial rating.

We received our B2 rating in January 1991 and at that time began to reexamine the issue of

---

[1]XENIX is a trademark of Microsoft Corporation.

[2]IBM is a registered trademark of the International Business Machine Corporation.

[3]RAMP is an acronym for Rating Maintenance Program. This is a program that allows a vendor to demonstrate that the rating of a product has been maintained across revisions.

extending the list of evaluated hardware bases. We established a Hardware Evaluation Process designed to demonstrate that the hardware bases in consideration were compatible with the IBM PC AT[4]. Used in this context "compatible" refers to more than simply being source code compatible. "Compatible" also means that all of the protection mechanisms provided by the IBM PC AT to meet the TCSEC requirements are also provided by the new hardware base. Any differences discovered between the new hardware base and the IBM PC AT must be shown to be security benign and not to affect the ability of Trusted XENIX to meet the TCSEC requirements. The specifics of the Hardware Evaluation Process are presented in following sections.

While establishing our Hardware Evaluation Process we presented our ideas to the Porting Working Group (now considered the RAMP Working Group) as well as NCSC Management. In addition we submitted a proposal to NCSC to evaluate 21 clones in a RAMP-like fashion. It was our suggestion that TIS perform the necessary analysis on the hardware bases in accordance with our Hardware Evaluation Process and that NCSC would review representative samples of our work in order to determine the validity of our process. It was our opinion that breaking down the work in this manner would alleviate concerns regarding how many machines were evaluated as the burden on NCSC resources would not increase based on the number of machines as NCSC would not be reviewing every piece of evidence generated for each machine.

In September 1991 we were assigned a team from NCSC of 3 evaluators to perform a "mini evaluation" of 6 out of the 21 clones. The 6 machines were chosen by NCSC based on NCSC marketing criteria. The scope of this Clone Evaluation included adding 6 new hardware bases as well as making software modifications to several device drivers.[5] The evaluation team agreed to basically follow the Hardware Evaluation Process proposed by TIS with some modification. The actual work break down did involve TIS performing the analysis for each hardware base but the evaluation team felt it necessary due to the unprecedented nature of the activity to review the evidence for each machine. In addition the evaluation team reviewed the results from our security test suites and performed a focused penetration testing effort. The rating on the revised version of the system, which included the software modifications and additional hardware platforms, was received in April 1992.

We are currently focusing on including the remaining 15 machines that have already been examined by TIS in accordance with our Hardware Evaluation Process just as was done with the 6 machines added to the EPL. We are confident that the remaining 15 machines could be addressed as a RAMP cycle. In addition to the original 21 hardware bases we examined, we also continue to perform our Hardware Evaluation Process on new machines. It is well known that an accreditor can make the decision to approve the use of an evaluated operating system on a hardware platform not included in the evaluated configuration. Thus, even if a machine is not

---

[4]Although the Hardware Evaluation Process can be used to compare any types of machines we decided to consider the IBM PC AT as our baseline for this initial endeavor.

[5]Although NCSC would only agree to include 6 machines in the scope of the evaluation we continued to perform our Hardware Evaluation Process on all of the 21 machines.

currently listed on the Evaluated Product List (EPL) we feel that our evidence can prove to be very useful to an accreditor for certification purposes.

The remainder of this paper provides an overview of the background behind the establishment of our Hardware Evaluation Process as well as describing the process in detail and provides a summary of our Clone Evaluation.

Background

Faced with the challenge of establishing what type of analysis was necessary to demonstrate compatibility we considered many difficult questions. What makes one hardware base compatible to another? Is a vendors claim sufficient? Or is it sufficient if one can show that the operating system runs successfully or that the security tests run successfully? What hardware interface attributes must remain the same? How should penetration testing be handled? Is penetration testing necessary?

In order to address these question thoroughly we decided to look at the bigger question, that is how do you make a modification to a B2 operating system and continue to maintain its rating. We were confident that after understanding this issue the level of work could be adjusted to suit the type as well as number of changes made. For instance a simple spelling mistake in an error message shouldn't require the same level of analysis as would a rewrite of memory management.

Changes to the TCB of an evaluated product can be classified into one of two major categories: those that cause the TCB to non-TCB interface to be modified and those that do not. In some cases this distinction may be difficult to make but the resulting benefit is that changes to the TCB that are not visible at the TCB interface may be easier to analyze.

For instance, if portions of the TCB can be modified and it can be demonstrated that the TCB interface has not been modified, then it can be asserted that the features provided by the original hardware and software continue to be provided by the modified hardware and/or software. In addition some of the assurances established in the original evaluation that involved analyzing the TCB interface, namely the Descriptive Top Level Specification (DTLS) (B2 and above) and/or the Formal Top Level Specification (FTLS) (A1), and possibly the security testing coverage would not have been affected by the modification and would not need to be readdressed. If the security test suite was developed through the utilization of a methodology requiring analysis of the TCB internals, such as the Grey Box Methodology[7], then its coverage would need to be reconsidered regardless whether the TCB interface had been modified. We are not stating that the security test suite should not be run unless the TCB interface is modified. In fact successfully running the security tests can validate a claim that the TCB interface has not changed. We are only indicating that in certain cases the security tests would not need to be reexamined for completeness unless the TCB interface were modified. In addition if the TCB interface has not been modified then one can assume that the definition of the model still holds true however previously established mappings may need to be reexamined to ensure their accuracy.

In terms of covert storage channels, changes that are not visible at the TCB interface can not

introduce new global resources that are shared among unpriviliged processes. However internal TCB changes may affect the manner in which previously identified shared global variables are used. Therefore one must demonstrate that changes made do not cause a previously identified shared resource to be used in a manner that would now be part of an illegal data flow. In addition one must examine the software modifications to ensure that any variables local to the software cannot be used as part of a covert channel.

The remainder of the assurances established in the original evaluation must always be considered to determine how they should be modified in order to continue to meet the TCSEC requirements. Of the these requirements, penetration testing requires some additional discussion. When performing an analysis of a small change to an evaluated product, we feel a focused penetration testing effort (versus a full scope effort) should be performed to ascertain that the system is still "relatively resistant to penetration". This focused effort should build upon the penetrations testing effort performed during the original evaluation. A full scope penetration testing effort is an event that should occur only on a predefined periodic basis, not as a part of every NCSC approved release of a product. It is common understanding that performing penetration testing does not guarantee that all problems within a product have been uncovered. It is simply an opportunity for an objective party to examine the system in an attempt to uncover subtle flaws missed by the developer. The appropriate level of effort for penetration testing should be determined based upon the number and magnitude of changes made to the TCB.

Hardware Evaluation Process

From the general problem of maintaining a B2 rating across new releases of the product, we extrapolated the conclusions that were pertinent when changing the hardware base. We defined "compatibility" in terms of two characteristics: (1) In order to be compatible, the hardware base in question must interact with the software in the same manner as the original hardware base (i.e. the TCB interface is not modified). (2) It must provide the same physical characteristics and be accompanied by a comparable set of system integrity tests. Differences in any of these areas must be shown to not be security relevant. Based on this definition we generated a Hardware Evaluation Process that defines the necessary steps to determine whether a new hardware base can be considered compatible to the IBM PC AT.

In order to demonstrate that the addition of a new hardware base does not modify the TCB interface, the devices within the new hardware base must be compared with the original devices, i.e. controller cards. An original device is defined to be any device found individually listed on the EPL for use with an IBM PC AT or found by default within the IBM PC AT. In terms of interaction with the software we feel a device, say HW2, can be considered compatible with HW1 if the following conditions hold:

a. The instructions provided by HW1 that are depended upon by the Trusted Operating System must be implemented in HW2. Differences in implementation (e.g., utilizing DMA capabilities, memory on board) must be shown to be invisible at the TCB interface.

b. It must be shown that additional instructions or capabilities provided by HW2 are not available

at the TCB interface.

In order to perform this comparison we obtained a solid understanding of the hardware interface characteristics provided by each original device as well as an understanding of the software that uses the device. In order to facilitate this information gathering, we created the concept of a Baseline Report. A Baseline Report contains a detailed description of the internals of the software interacting with the device, the software interface to the kernel, the software interface to the hardware, and a complete description of the hardware interface. In terms of the hardware interface description, we indicate which features, registers, commands, provided by the hardware are used and not used by our software. These Baseline Reports have become part of our design documentation. They are considered Configuration Items (CI) and therefore changes to them will be tracked via Configuration Management (CM).

From these baselines we extracted a template of hardware interface characteristics. A two step process was then used to determine if the candidate device meets the two requirements described above. The first step involves examining each candidate device to determine whether it differs from the template in any way. This examination utilizes manufacturer supplied information as well as internal analysis and penetration testing when necessary. All differences between the candidate device and the template are noted. The second step involves the analysis of the various identified differences. When a difference is noted, the information from the Baseline Report is used to determine if the difference is visible at the TCB interface. If the difference is visible at the TCB interface a recommendation is made for how to address the difference. If it is determined that a device cannot be supported without causing the TCB interface to be modified, then the device driver will need to be examined. It may be determined that the device driver could be modified in such a way that supporting the device does not change the TCB interface. This software change will then need to be validated according to the our CM Process. The two steps, examination and analysis, are performed by different individuals.

All devices that map into I/O address space must be examined via this method. This can include DMA controllers, interrupt controllers, real time clock, timer, hard disk controllers, floppy disk controllers, video controllers, parallel port controllers, serial port controllers, and keyboard controllers. Devices that do not map into I/O address space, namely memory, hard disk drives and floppy disk drives, need not be examined in this manner, as their functionality is sufficiently tested via the execution of our Security Test suite. Any additional functionality provided by these devices is not accessible to the user.

The initial set of 21 additional hardware bases all utilized either Intel 80286 or 80386 CPUs and in some cases Intel 80287 or 80387 coprocessors. Thus no analysis of these components was necessary as they are considered original devices.

Evidence generated for each individual hardware base includes a list of the devices included within each clone indicating the manufacturer name and model or revision number of each device. For each device not currently included in the EPL list of supported hardware, a template will be completed comparing the candidate device to the original device. These templates are maintained as evidence.

The device comparisons described above serve as a means to demonstrate that the new hardware is compatible with the original hardware in terms of interaction with the software. In addition the physical features of the new hardware must be examined and compared to those found in the original hardware. The capabilities and usage of each of these features are described in detail in a document we refer to as the Summary of Evidence. The user's manual associated with the hardware base normally provides a bulk of this information. Features that need to be explored include system initialization, booting capabilities, passwords, setup programs, keylocks, special key sequences, etc. The goal of this activity is to identify how the physical features of the new hardware base differ from the physical features of the original hardware and to modify the Trusted Facility Manual (TFM) for Trusted XENIX as necessary to support the new hardware bases. From the Summary of Evidence we create an entry for the specific hardware base for our TFM. In general the TFM must describe how to prevent unauthorized access to the internals of the workstation and to prevent users from interfering with booting Trusted XENIX securely. In addition if the new hardware base provides a CMOS setup program that might provide a user with the ability to modify the system date and time, then access to the setup program must be prohibited. Furthermore the new hardware base may provide additional features that are security relevant but have not been previously dealt with in the TFM. The TFM must also describe how access to these features is denied.

In addition the Power On Self Tests (POST) and advanced diagnostics provided with the new hardware base must be examined to determine whether they are sufficient to meet the system integrity requirement as were the diagnostics provided by the original hardware platform. Finally the security test suite must be run on each new hardware base to further support our argument that the TCB interface was not modified.

Clone Evaluation Process

Summary of Changes

In September 1991 we were assigned a team of evaluators from NCSC who had been tasked to evaluate six of the many hardware platforms that we had examined. Basically the inclusion of the new hardware bases involved changes to the hardware layer of the TCB. In addition though the hardware dependent software layer of the TCB was also modified to include three new device drivers to support SCSI hard disk controllers. Our initial assessment of these changes was that they would not cause the TCB interface to be modified.

While performing the necessary actions to demonstrate that the interface had not been modified by the changes described above, we determined that there were some additional modifications that should be made to the existing keyboard and video device drivers. The changes to the keyboard driver remained invisible at the interface and were handled just like the addition of the new device drivers. However, the changes to the video device driver were visible at the TCB interface, and were treated in a slightly different manner.

The changes to the hardware layer were handled in accordance with our Hardware Evaluation

Process. The following subsection describes how we addressed changes to the hardware dependent software layer.

Hardware Dependent Software Layer

Changes to the hardware dependent software are handled relatively in the same fashion as they are at the hardware layer although the type of analysis and evidence generated changes. The first step involves understanding the change to be made, its ramifications to the product, and then implementing it in a controlled fashion. As part of this step we determine whether the TCB interface has been modified and, if not, demonstrate that. The second step involves performing additional security analysis activities in order to continue to satisfy the TCSEC requirements.

The first step of understanding and correctly implementing a change at the hardware dependent software layer revolves around our CM process. All additions or modifications to the hardware dependent software must be examined according to the CM process to ensure that trusted software is being developed consistent with the policy model, DTLS, security tests and design documentation and the security policy of the system is not subverted.

Our CM process encompasses a set of activities specific to the development of new software. New designs and code must examined through a series of reviews before they will be made new CIs and included in the CM Library. Once a CI has been included in the CM library, changes to it will be processed according to the maintenance phase of the CM process, which also involves several reviews.
As part of the CM process, it is determined whether the TCB interface has been modified and, if so, then the existing design documentation and security test suites are appropriately modified.

Evidence generated from this first step includes the design documentation, unit test documentation, minutes from review meetings, and CM process documentation. In addition, the entire Security Test suite, including any new changes made, must be successfully run as a final step in either approving the addition of a new CI or the modifications made to existing CI's. Following the reviews included in the CM process and the actual implementation of the change, we address the modeling and covert channel requirements and perform necessary analysis to demonstrate that these requirements are still met.

Requirements

The following section summarizes how the B2 TCSEC requirements were addressed for the Clone Evaluation.

Security Policies, Accountability A detailed description of the changes made to the existing system was provided. The changes were confined to minor modifications to existing device drivers and the addition of three new device drivers. These changes were easy to understand and resulted in minimal change to the TCB interface. The code affected by these changes is referred to as the I/O subsystem. This subsystem is not responsible for explicitly implementing any of the required security policies thus the changes have not affected the systems ability to implement the required policies.

System Architecture Same assurances from the original evaluation hold true.

System Integrity For the currently supported system configurations, this requirement is fulfilled by the power on self test (POST) and a set of advanced system diagnostic tests associated with the IBM PC AT or IBM PS/2. Although most vendors provide similar test packages, we have found that the hardware protection mechanisms of the CPU are not sufficiently exercised and in some cases, the tests simply are not comprehensive. Thus, we have created our own set of tests to exercise the hardware protection mechanisms. In addition, we have decided to require that a suite of diagnostics generated by a third party, Checkit V3.0 provided by Touchstone Software, be obtained in order to satisfy the system integrity requirement.

Covert Channel Analysis The changes to the hardware layer are not visible at the TCB interface. Thus, no new global resources shared among processes could have been introduced. However, the addition of new hard disk device drivers could potentially introduce new local shared variables or cause already identified shared variables to be used as an illegal channel. Therefore, we reviewed the drivers to verify that they introduced no new illegal flows and to ascertain that the shared variables identified in the original evaluation via the review of the original hard disk driver were used in the same manner. Finally, although the change to the video device driver was indeed visible at the TCB interface, we demonstrated that it could not introduce any covert channels, since Trusted XENIX does not allow processes at different security levels to have simultaneous access to the console. Our covert channel bandwidth analysis was performed on all of the machines and a report providing an worst case analysis of the results was generated. The worst case scenarios were all found to be acceptable to the evaluators.

Trusted Facility Management Same assurances from the original evaluation hold true.

Security Testing The changes to the hardware dependent software involved changes to the I/O subsystem, i.e., device drivers. The I/O subsystem performs no security relevant events and is not tested as part of the security test suite. Thus the coverage provided by the original security test suite is still sufficient.

The complete suite of security tests was run on each new hardware base adding further assurance that the TCB interface has not been affected by the modifications made to the TCB. In addition our evaluation team performed a penetration testing effort focusing on the peculiarities of each system, i.e., setup programs, system passwords, etc.

Design Specification and Verification No new subjects or objects were introduced, nor has the way in which existing subjects access existing objects been modified, thus the assurances from the original evaluation still hold.

Configuration Management The CM Plan has been broadened in order to address developing new software to be integrated into Trusted XENIX. The CM process will be invoked for every change made. Changes to the hardware layer will entail adding the new hardware bases to the list of hardware maintained in the CM Library. In addition a relationship was established between the hardware vendor and TIS by which TIS will be informed of changes made to the supported product line.

Security Features User's Guide Not modified.

Trusted Facility Management All supported hardware bases must have a section in the Trusted Facility Manual describing physical security issues particular to the machine. These new entries must at a minimum address the following issues.

Special Features: Any special features of the system (e.g. special key sequences, external buttons, keylocks, passwords, etc) must be described.

Identify Booting Capabilities: A description of the bootable devices will be provided. A discussion will be included describing how the boot devices can be protected.

Discussion of BIOS: A discussion of the BIOS utilized by the hardware base will be included listing the functionality provided.

Test Documentation As the security test suite was not modified, the test documentation remains unchanged.

Design Documentation Design documentation for each new hardware base has been obtained. The Kernel Architecture Document does not currently describe in detail the various device drivers. Thus, it did not need to be modified as a result of these changes. However the design documents created as a result of the CM development activities will be considered part of the Kernel Architecture Document. As the addition of the new device drivers and the change to the keyboard device driver are not visible at the TCB interface they did not result in any changes to the Descriptive Top Level Specification. The change to the keyboard driver however is visible at the interface and the DTLS was modified to reflect that change.

Conclusions

At the Sixth RAMP Workshop in Los Angeles a draft set of requirements for RAMP at B2 and above were distributed. These requirements allow for RAMP at the higher levels to be performed by a variety of different personnel. One of the options includes the combination of the vendor and 3 to 4 NCSC evaluators, and is expected to take 3 to 9 months. Our Clone Evaluation effort exemplifies this option. The only differences are that we did not have the opportunity to present our approach to a Future Change Board [6] nor were we able to be present at the final TRB. Although both of those actions would have proven beneficial, we feel that the success of our endeavors provides realistic evidence that RAMP can work with the higher level products. In some instances we feel that performing a RAMP can be easier on a higher level product as the system is better layered, more modular and better understood, thus the ramifications of a change can more easily be determined and addressed.

---

[6]The Future Change Board will consist of TRB members, the Chief Evaluator, and other members of the NCSC evaluator community who have worked with the product.

Our experience has demonstrated to us that RAMP should be robust enough to address changes to any of the operating system layers. The type of analysis performed and the evidence generated will differ across the layers but the basic philosophy of addressing change should not.

It is essential that RAMP become available for all levels of systems immediately. Vendors at all levels will always need to make changes to their evaluated products, either to add enhancements, support new hardware or make corrections. Time is of the essence especially in terms of supporting new hardware due to its limited lifespan. Once a vendor has made the decision to support a new hardware base and has performed the necessary analysis there must be a mechanism by which the hardware base can be included in the evaluated configuration before the hardware base becomes obsolete.

In conclusion we found our experience enlightening. We are currently applying our approach to RAMP with several software applications as well as additional hardware platforms and are anxious to embark in a recognized RAMP activity.

References

[1] DoD Trusted Computer Systems Evaluation Criteria, DOD 5200.28-STD, National Computer Security Center, Ft. Meade, MD, December 1985.

[2] "Trusted XENIX Product Evaluation Bulletin," Report No. CSC-PB-004-87, National Computer Security Center, Ft. Meade, MD.

[3] National Computer Security Center, "Rating Maintenance Phase: Program Document," NCSC-TG-013, June 23, 1989.

[4] D. Bell, G. Benson, T. Redmond, D. Sterne, "Trusted Reuse Issues," Internal TIS Report Number 347, August 24, 1990.

[5] Richard R. Linde, "Operating System Penetration," *Proceedings of the National Computer Conference*, 1975.

[6] Richard A. Kemmerer, "The Shared Resource Methodology: An Approach to Identifying Storage and Timing channels," *ACM Transactions on Computer Systems*, 1(3):256-277, August 1983. University of California, Santa Barbara.

[7] "A Guide to Understanding Security Testing and Test Documentation," National Computer Security Center, NCSC-TG-023, DRAFT.

Finding Security Flaws in Concurrent and Sequential Designs using Planning Techniques

Deborah A. Frincke [frincke@cs.ucdavis.edu]
Myla Archer [archer@cs.ucdavis.edu]
Karl Levitt [levitt@cs.ucdavis.edu]

Division of Computer Science, University of California, Davis

Abstract

This paper[1] presents an automated system (SPLAN) that can assist in the validation of secure systems. SPLAN, based on classical planning ideas, takes as input a system description (specifications and/or code, including concurrent programs) and a more abstract specification (e.g., a specification of disallowed states based on a security policy) and attempts to generate a sequence of operations or code statements that will cause the system to reach a state disallowed by the policy. Thus SPLAN attempts to generate sequences of operations that violate the security policy. SPLAN has built-in heuristics to reduce the space of operation sequences it searches, such as loop detection, templates of operation sequence schemes likely to expose flaws, and operations that are flagged as suspicous by a security flow analyzer. We believe that SPLAN would be most useful in the validation process when applied after the use of conventional testing and of a flow analyzer but before verification is attempted. This paper presents various examples showing how SPLAN can detect covert pictorial channels in a specification for a secure user interface management system and in an erroneous—and previously published—mutual exclusion program.

1 Introduction

The Orange Book [2] indicates a number of approaches in the validation of secure systems, including the verification of specifications with respect to a security policy (for A1 certification) and testing (e.g., of program code with respect to specifications). A technique that satisfies the Orange Book's requirements for A1 certification is to use a ***security flow analyzer***, for example see [4][15]. Briefly, a security flow analyzer considers each operation specification in isolation, and determines if the specification has the potential of contributing to an information flow disallowed by a security policy. The aforementioned flow analyzers will detect access control violations as well as covert channels. Furthermore, flow analyzers carry out essentially syntactic checks on the system description and, hence, perform quite well even on large descriptions.

There are several drawbacks to these flow analyzers. First, they erroneously flag many operations as insecure: the operations cannot be invoked with the inputs necessary to produce the disallowed flow, or they cannot be made visible to a user. In simple terms, the flow analyzers produce a pessimistic security analysis of a system; the lower the level of the specification, the more pessimistic the analysis. Second, most systems exhibit disallowed flows that cannot be removed. The security policy mechanized by the flow analyzers is too strong.

With regard to the first limitation of flow analysis, a comment by by Gasser [9] is relevant:

> Because the syntactic flow analysis technique only flags potential flow violations, additional covert channel analysis is required to determine whether the violations are real. There are no tools that help you to do this, since it requires looking at the specification as a whole and deducing or proving additional properties. A typical argument to support the contention that a flow is not real would be based on the fact that the specification lacks certain functions that could exploit the flow.

Inspired by Gasser's challenge, we have developed a tool, called SPLAN, that attempts to determine if a flow is real. SPLAN accepts as input a description of a system (a specification of its operations and/or program code, including concurrent code) and a security policy (in the form of program states disallowed by the policy, e.g., there is no flow except from a user at a low level to a user at the same or higher

[1]We gratefully acknowledge the support of D. Mansur (Project Manager, Lawrence Livermore National Laboratory, 442423-25173) and E. Siarkiewicz (Project Manager, Rome Laboratory, F30602-88-0-0025).

level. Of course, it is necessary to define what "user" means, which in our case is the process (including its memory and registers) working on behalf of a user. From this input, SPLAN attempts to generate a sequence of actions (operations or program statements) that will cause the system to reach a state in which the policy is not satisfied.

SPLAN is implemented in Prolog and based on classical planning methods, such as those implemented in STRIPS [12]. Such methods are now in disfavor, primarily because they explore many unproductive plans before (if ever) hitting on a plan that works. In our application, particularly when dealing at the level of concrete program code, many unproductive sequences would be explored. To improve on the performance of the STRIPS generation of planners, SPLAN can apply domain-specific knowledge. The knowledge it uses includes:

- Operations that must be present in any generated sequence. For example, a flow analyer will identify suspicious operations, operations that SPLAN will include in any sequence it generates.
- Templates indicating likely actions that can produce disallowed states. For example, all successful attempts to exploit covert channels involve the actions of two or more processes separated by one or more context switches.
- Loop detection, to prevent the exploration of nonterminating sequences.
- Distance measures, to assess how far a state is from the goal state (where the disallowed flow is consummated).

As motivated above, SPLAN would find use in determining if formal flow violations detected by a flow analyzer are real; the security policy in question is, typically, the ***mandatory security policy***. However, SPLAN can be programmed to detect violations of other security-related policies. For example, it can be used to analyze the code of an operating system to identify sequences of user inputs that could cause a system's authentication mechanisms to be bypassed. The domain-specific knowledge that would reduce the search space could be that a successful sequence of actions would include an interrupt after a variable has been checked with respect to a particular property.

In general, we anticipate that SPLAN-like methods would be used between testing (where obvious bugs are discovered) and verification (where the system is verifed with respect to a ***realistic*** policy – e.g., a policy less restrictive than that mechanized in flow analyzers). Table 1 summarizes the features of testing, verification and SPLAN with respect to their use in connection with secure systems.

To illustrate the usefulness of SPLAN, this paper provides three examples. The first two involve resource management systems (specifically, user interface management systems) and the third involves access control between simultaneously executing processes. The area of user interface management was selected for the first two examples because it is of interest to explore security problems associated with such systems. In particular, the user interface manager necessarily has access to all of the graphical interface objects that depict or manipulate data, and is therefore capable of illicit information transfer between users. Furthermore, with the addition of a graphical component to an application, a new group of channels—***pictorial covert channels***—become possible. These channels either (1) occur through the use of views that are based upon inappropriate data, (2) are part of shared resources within the underlying user interface management system or (3) rely upon interactions between applications, through inappropriately captured pictures or events.

The third example combines access control and simultaneously executing processes. Development of correct algorithms for the synchronization of concurrently executing processes can be difficult; there have been several incorrect algorithms published. Potentially, one must consider all possible interleavings of process execution before the algorithm may be said to be correct. Most concurrent program debuggers have been developed in order to assist the programmer once an error has appeared, rather than to detect potential problems. Some debuggers can detect potential race conditions; unfortunately, these debuggers frequently provide so many false positives that their value is greatly reduced[11]. Synchronization algorithms are relevant to the development of secure systems because unexpected interleavings of operations are a common source of many security flaws.

In the remainder of this paper, Section 2 discusses some related techniques. Section 3 describes SPLAN, Section 4 gives an overview of SPLAN's implementation, Section 5 provides three sample flaws that may be found using this method, and Section 6 outlines the work in progress.

2 Related techniques

Testing: Other authors have studied methods for exposing flaws in software. Typically, as in [13], dataflow analysis techniques have been used to study sequential programs. Taylor [18] has extended existing techniques to concurrent programs, emphasizing

	Verification	Testing	SPLAN
Objective	Try to prove correct in general w.r.t. spec	Try to show correct/incorrect for certain input	Try to show incorrect w.r.t. common flaw classes
Upon Error	Shows what cannot be proven	Shows improper behavior for some input	Shows an execution path exhibiting flaw
Completeness; What do the results mean?	When successful, proves all cases correct w.r.t. spec	Shows behavior correct for some input set	Shows a flaw or set of flaws cannot occur if terminates
User skill/automation	High, considerable effort but many tools	Varies; often requires high familiarity with code	Medium; must know how to specify flaws and templates if used
User provides	Specification, code, property to verify	Executable spec or code and test cases	Spec or code, flaw description
Position in Cycle	Last	First and throughout	After some testing, before (possibly during) verification
Value of Partial Steps	Low unless complete	Each test supplies some information (incremental)	Each flaw check provides some info (incremental)
Domain Knowledge	"Theories" about security	Path testing, data flow analysis (incremental)	Essential; e.g., suspicious operations from flow analysis
False Positives,	None for code or spec if complete	None (may not be repeatable)	None, but may not halt
False Negatives	Can happen; e.g., proof is weak	No guarantee; cannot usually test all cases	Not for flaws checked (completeness)

Table 1: A Comparison of Validation Techniques.

detection of parallelizable code segments with special attention to Ada. Knowledge-based techniques have also been applied to the problem of debugging. Seviora [17] identifies kinds of knowledge that a debugger could use; for example, knowledge about what a program should do and should not do, likely flaws (especially in concurrent programs), and the granularity of testing. The tools surveyed do not attempt to automatically generate test cases. A more recent knowledge-based debugging system is described in [19], which uses a knowledge base to reduce the data from a debugging session to allow for more easily understood replays. Our tool is more flexible than the conventional approach of using test data; it can find general classes of data (i.e., detect sets of flaws), and may also be used to decide when particular execution paths lack specified security flaws.

Planning: In [1], Feigenbaum and Barr describe a plan as "a representation of a course of action." In this paper, planning techniques are used to develop a course of action (sequence of instructions) that will transfer information from one user to another. If this information transfer is illegal with respect to the desired security policy of the system, then the plan identifies a security flaw in the system.

Many different planning techniques are discussed in some detail by Wilensky [20] and Nilsson [12]. One simple technique used by many planning systems is *forward chaining*. In forward chaining, a system first starts with an initial state, a collection of goals G_i to be achieved, and a collection of actions A_i that may be used to achieve them. These goals may have varying importance. Further, it must be possible to examine two states and determine how 'close' they are to one another. This examination requires some reasonable way of measuring distance between states; this metric will depend upon the components of a state and may change considerably between applications.

Secure UIMS: The Compartmented Mode Workstation program, CMW, is the Defense Intelli-

gence Agency's trusted computer systems criteria for a secure X system. A few vendors, including SUN and SecureWare, have developed versions of X meeting these criteria. The issues involved in this effort are summarized by Epstein and Picciotto in [3]; Trusted X systems attempt to 'graft' security onto systems that lacked nearly all security features; X was actually designed deliberately to avoid enforcing security policies of any type, and contains many mechanisms to promote sharing between applications.

X is based upon a client/server model, with the server managing the window manager and clients (applications) that send requests to manipulate resources, etc.) [16]. X provides minimal protection; the X server only determines whether or not a particular client may be connected. All connected clients are treated equally; in fact, connected clients may even turn off the requirement that the X server perform any checking when future clients request that they be added to the system. The window manager's job is to manipulate windows upon the console screen. X does not offer privileges of any type, so the window manager is just another client of the server. Our example illustrates some of the security liabilities of unrestricted access to graphical systems.

3 A Prototype Testing System

SPLAN, written in Prolog, is based upon the methodology used in two earlier systems: TPLAN [7] and CTPLAN [8]. TPLAN was developed specifically to detect security flaws in operating system specifications. TPLAN represents the operations of the system being tested as STRIPS-like rules [5]. The operations can represent the system at any level of abstraction, ranging from specifications of the operations visible at the system interface to statements in a programming language. Further, the representation can be in terms of more than a single abstraction, i.e., a combination of specifications and executable code. TPLAN has been used primarily to identify security flaws in simple operating systems, the systems being represented abstractly in terms of formal top-level specifications. CTPLAN also represents algorithm statements as STRIPS-like rules. CTPLAN can detect a variety of flaws in concurrent algorithms, including deadlock and mutual exclusion. CTPLAN has been used successfully to detect flaws in algorithms that had actually been published, e.g. in [10].

SPLAN combines these two prototypes, producing a system that can detect sequences that violate an information flow policy or mutual exclusion violations that may be either within algorithms or within operation specifications.

Use of a system such as SPLAN is of the most benefit when the programmer is faced with the task of determining whether or not a software system contains a specific behavioral flaw. When dealing with certain flaws, such as synchronization of processes and information flow between processes, it is often easier for the programmer to state what should *not* happen, rather than what should happen. This specification of an undesirable situation is the goal which SPLAN uses to construct a plan. For example, permitting two processes to manipulate the same object at the same time is usually undesirable. A manifestation of this flaw (a violation of mutual exclusion) may be described easily:

$$\neg((\text{Process 1 modifies } O \text{ at } t) \cap (\text{Process 2 modifies } O \text{ at } t))$$

However, the algorithm that actually prevents this from happening is much more difficult to state. Furthermore, the granularity of the algorithm's encoding may also affect the presence of a flaw, as well as SPLAN's ability to detect it.

This paper represents characteristic properties such as mutual exclusion and information flow as predicates that become the goal for a planner. Additionally, certain heuristics that reduce the search space, such as loop detection are identified.

Using operation specifications and a description of a particular type of information flow, SPLAN attempts to find a sequence of operations (a plan) that produces the specified flow. Information flow is described by exhibiting an initial—valid—state and a final state wherein the user has access to unauthorized information. SPLAN attempts to produce a flaw-illustrating plan if one exists. This is in contrast to flow analyzers [4], as not all the channels these systems identify actually permit information flow [6], and exhaustive checking all sequences is not feasible. Use of SPLAN permits the user to focus on actual flaws in the system at the specification stage. A side benefit of the planning approach is that it produces a general test sequence, that is, an expression that subsumes many cases of input values that cause the flaw to be revealed. For a class of covert channels, an approximation to the bandwidth can be derived using an appropriate expression.

4 Implementation of SPLAN

There are four types of rules within SPLAN :

- **Architecture/Difference:** These rules define the state; for example, the system widget list in a UIMS.
- **Planning:** These rules use heuristics to examine the algorithm rules and produce a plan to reach the goal state containing the desired flaw.

```
[user-observed-value,
  [list of users [name]],
  [symbol table, application [name, application, value],
   symbol table, display [name, value]],
  buffer,
  [ [list of application object differences],
    [list of display object differences]]
  [ intermediate plan ] ]
```

Figure 1: X11-style UIMS Architecture

- **Algorithm/Operation:** Algorithm rules embody the SPLAN translation of the algorithm to be examined; operation rules describe operations using preconditions and postconditions.

- **Input:** These rules define the flaw to be examined, and the initial state of the system (including the number of processes executing).

In addition to producing test plans, SPLAN may be used to symbolically simulate the execution of a series of statements and to test the validity of a sequence of statements. When provided with an initial state and a test sequence of statements, SPLAN will produce the state(s) that will result if they are executed, provided that the test sequence is valid.

Architecture Rules: Architecture rules describe the components of a system. For example, Figure 1 shows a sample architecture rule for a typical user interface management system. Architecture rules modify SPLAN's view of the system state. They may also be considered predicates to be instantiated or revoked depending upon the operation applied. For example, in a UIMS having multiple buffers, there exist architecture rules allowing SPLAN to observe and modify buffer values within the current state. Alternately, one may consider the system to contain predicates such as "Buffer I of user A has value X" and "Buffer I of user A is modified to contain value Y." Architecture rules are used within operation rules to describe preconditions and postconditions that determine whether an operation is executable under the current conditions.

Difference rules are used to detect the differences between an initial and goal state, and to set up as subgoals the elimination of these differences. If process A's application variable $Var(A, i)$ contains the value X in the initial state and the value Y in the final state, then SPLAN adds the subgoal *object difference* [[A, i, X], [A, i, Y]] to the goal list (widgets are objects). Every state component has its own collection of difference rules, since the structure of these depends upon the component's representation.

Planning Rules: Planning rules are used to manipulate plans. There are two types of planning rules: those that eliminate 'unnecessary' goals, and those that select the subgoal to be achieved next. Unnecessary goals are goals already achieved (as a side effect of solving other goals), or goals that do not cause any real change in state. The system may have two goals: causing user A's application variable i ($Var(A, i)$) to contain value X (Goal 1), and causing user A to become the currently active user (Goal 2). Suppose that in the initial state, user B is active, user A is blocked, and only active processes can modify objects. Further suppose that the system chooses to work on Goal 1 first, and achieves it via the following plan: (1) Make user A active (Subgoal 1), (2) Write X into $Var(A, i)$ (Subgoal 2). Step 1 also achieves Goal 2. Steps 1 and 2 accomplish *subgoals* 1 and 2 of Goal 1.

The second type of planning rule determines the goal SPLAN will try to achieve first. In theory, the goals are achievable in any order: if SPLAN determines that it is impossible to achieve all goals following a particular order, it backtracks and tries them in a different order. However, this is not always successful, since SPLAN does not recognize all types of infinite loops in planning sequences, though exact duplication of states are recognized. SPLAN may attempt to achieve a sequence of goals where the solution to the first goal 'undoes' the solution to the last goal. This is illustrated by the following example: Consider a system containing Goal 1 (described earlier) and Goal 2′: make user B active. This time, user A is active initially. The following sequence will loop infinitely: (1) Make user B active (to achieve Goal 2′), (2) Make user A active (to achieve Subgoal 1; this unfortunately undoes Goal 2′) (3) Make user B active (to achieve Goal 2′). Both subgoals could have been achieved if SPLAN had completed both of the Goal 1 subgoals before attempting to achieve Goal 2′. To avoid this type of looping, SPLAN uses two heuristics: complete all the subgoals of a goal at one time, and complete the most complicated goals first, since these are the goals most likely to undo other goals. Goal complexity is measured by counting the number of subgoals it contains. These two heuristics are insufficient to prevent all infinite loops, so SPLAN contains a simple form of loop detection to identify repeated states. This is a common problem in planning systems [12].

SPLAN's planning rules govern the way in which tests that expose algorithm flaws are found:

1. SPLAN searches the Algorithm Rules to find a statement[2] within a process that can either im-

[2] A statement is an invocation of one of the specified operations.

mediately eliminate a difference, or, if it cannot, can potentially lead to a statement that can eliminate a difference (these are found by backtracking through the algorithm execution steps).

2. SPLAN next checks to see if the statement can be executed through to completion.

3. SPLAN then looks to see if execution of the statement would duplicate a system state exactly. Since SPLAN permits looping, this is necessary to eliminate infinite attempts to execute the same series of statements. It will also produce shorter test plans than if states are permitted to repeat.

4. Steps 1-3 are then repeated, until all differences have been eliminated.

SPLAN searches for 'forward differences' when detecting flaws in algorithms, and 'backward differences' when detecting flaws in operation specifications. In general, one has more information about the starting state of an algorithm than about the final state of an algorithm; operations have no explicit sequence of steps and thus no 'required' initial state. Most algorithms have definite specifications about the starting state of the variables involved, such as semaphores. This information is not readily available (or necessary) for the final state, since it is often the incorrect usage of these semaphores and global variables that results in the flaw that is to be detected.

Input Rules: Input Rules define the initial and final state of the system. Each state describes the following: the state of the system variables and semaphores, the processes that will exit, and the code each process executes. SPLAN's purpose is to determine the sequence of statements that will transform the initial state into the final state. Figure 5 shows an initial and a final state used in a later example. If a variable has value `dontcare`, then SPLAN will modify it as needed.

Algorithm/Operation Rules SPLAN permits systems to be described either in terms of algorithms or operation specifications. Algorithm Rules are used to encode the algorithms to be tested. Figure 3 gives an example of the rules that must be defined for each algorithm statement. The rules have the following form:

changes(*Algorithm-name, Process-identifier, Type-of-state-object, Name-of-state-object*)
prestatement(*Algorithm-name, Current-statement-number, Previous-statement-number*)
statement(*Algorithm-name, Process-identifier, Current-statement-number, Incoming-state, Outgoing-state*)

where *Current-statement-number* is the step in the algorithm, *Type-of-state-object* describes the type of object modified by the statement, *Name-of-state-object* is the actual name of the object changed, *Previous-statement-number* lists the possible preceding statements, *Process-identifier* is the actual process executing the statement, and the rest are self-explanatory.

The way in which algorithms are encoded for SPLAN has an enormous effect on the type of flaw that may be detected. In particular, certain flaws will only be detected if the statements are translated with a fine-grained level of atomicity, and others will be more easily detected with a coarse-grained level of atomicity, due to reduced search time.The effects of granularity refinement are discussed in more detail in [8]; at present, the user is responsible for encoding algorithms at the proper refinement level.

SPLAN uses a Pascal-like minilanguage to describe algorithms. The only specialized software support for synchronization is the semaphore. This structure was included so that algorithms that use such structures could be easily implemented and processed; in addition, other specialized language structures used for synchronization (such as monitors and conditional critical regions) may be readily implemented using semaphores. **Test-and-Set** and **Swap** are included because these instructions are fairly typical of the type of hardware level support provided for synchronization; they are defined to be atomic [14]. Once an algorithm has been described in this fashion, it is translated into Prolog statements such as the one in Figure 3, which shows the translation of a simple flag-setting statement. This language subset is sufficient to describe a wide range of algorithms. However, it is often necessary to implement certain language features in terms of these atomic statements, which may affect the flaws that can be detected.

5 Examples

This section provides three examples of flaws that may be discovered using SPLAN. Two of them involve flaws within user interface management system specifications: a blatant flow of information through a common cut and paste buffer, and a more complex form of covert flow. The third describes a mutual exclusion flaw that violates access control rules within

an operating system that permits concurrency.

5.1 Simple Information Flow

This section describes the path that SPLAN follows to come up with a very simple example of information flow within a standard X11-like UIMS using a single cut and paste buffer. The operations available to SPLAN include some standard operations for reading and writing to specific memory locations, and two more added to simulate the standard buffer in a user interface management system: *Cut, Paste.* The system buffer *Buffer* is read and written using these new operations. Clearly, flow between users may occur via this buffer. As a simplification, the system is assumed to contain only two users, each with exclusive access to two blocks of memory and having local program variables Var_i. The following are the initial and final states:

5.2 A more subtle example of insecure flow

Initial State:

$$\left(\begin{array}{l} View(U_1,i) = \text{TOPSECRET-VAL} \\ Level(U_1,i) = \text{TOPSECRET} \\ View(U_2,j) = \text{SECRET-VAL} \\ Level(U_2,i) = \text{SECRET} \\ Var(U_1,a) = < nil > \\ Var(U_2,b) = < nil > \\ \\ \text{Buffer} = < empty > \end{array}\right)$$

Final State:

$$\left(\begin{array}{l} View(U_1,i) = \text{TOPSECRET-VAL} \\ Level(U_1,i) = \text{TOPSECRET} \\ View(U_2,j) = \text{¡undefined¿} \\ Level(U_2,i) = \text{SECRET} \\ Var(U_1,a) = < nil > \\ Var(U_2,b) = TOPSECRET - VAL \\ \\ \text{Buffer} = < undefined > \end{array}\right)$$

SPLAN begins with the goal of finding a plan whereby one user obtains information originally contained in the second user's memory. This is stated by defining an initial state where $User_2$'s block does not contain $User_1$'s information, and a goal state where $User_2$'s block does contain $User_1$'s information.

1. $Mem(User_1,k)_0 \neq Mem(User_1,k)_0 \neq 0$
   $Mem(User_1,k)_t = Mem(User_1,k)_t \neq 0$
   Using $User_2$ as the initially active process and $User_1$ as the active process in the goal state shortens the plan, though this is not required for a correct plan.
   Possible Plans for Goal 1:
   - $Store(i,k)$ by $User_1$, with $Var(User_1,i) = Mem(User_1,k)_0$

```
changes(incorrectPc1, 1, variable, flag1) .
changes(incorrectPc1, 1, pc, 1) .
prestatement(incorrectPc1, 1, 0) .

statement(incorrectPc1, Id, 1,
    state(Semaphores, Progvars, Progcounters),
    state(Semaphores, NewProgvars, NewProgcounters)) :-
lookupSymtab(Progcounters, [incorrectPc1, Id], 0),
        updateSymtab(Progvars, [flag1, true], NewProgvars),
        updateSymtab(Progcounters, [[incorrectPc1, Id], 1],
                     NewProgcounters) .
```

Figure 3: Translation of `flag1 = true`

   - $Purge(User_1,k)$)

   Both modify memory; however, *Purge* can only write 0 and is thus not useful for transferring arbitrary information directly.
   Choose plan *Store(i,k).*

$$\overbrace{\ldots\ Store(i,k)}^{User_1}$$

$$\left(\begin{array}{l} View(U_1,i) = \text{TOPSECRET-VAL} \\ Level(U_1,i) = \text{TOPSECRET} \\ View(U_2,j) = \text{TOPSECRET-VAL} \\ Level(U_2,i) = \text{SECRET} \\ Var(U_1,a) = < nil > \\ Var(U_2,b) = TOPSECRET - VAL \\ \\ \text{Buffer} = < undefined > \end{array}\right)$$

   This sets up a new goal:

2. $Var(User_1,i) = Mem(User_1,k)_0$ Possible Plans for Goal 2:
   - $Cut(i,k)$ with t=0
   - $Cut(i)$ with $Buffer = Mem(User_1,k)_0$

   If *Cut* is chosen, SPLAN must satisfy the precondition that there some object within $User_1$'s object set contains the value in $User_2$'s block. For illustration, *Cut* will be chosen.
   Choose plan *Cut(i).*

$$\overbrace{\ldots\ Cut(i)\ Store(i,k)}^{User_1}$$

$$\left(\begin{array}{l} View(U_1,i) = \text{TOPSECRET-VAL} \\ Level(U_1,i) = \text{TOPSECRET} \\ View(U_2,j) = \text{TOPSECRET-VAL} \\ Level(U_2,i) = \text{SECRET} \\ Var(U_1,a) = < nil > \\ Var(U_2,b) = TOPSECRET - VAL \\ \\ \text{Buffer} = TOPSECRET - VAL \end{array}\right)$$

   This sets up a new goal:

3. $Buffer = Mem(User_1,k)_0$ Only a *Paste* can cause *Buffer* to contain the desired value. The current process cannot do the write, since it does not have $User_2$'s information. Thus, $User_2$

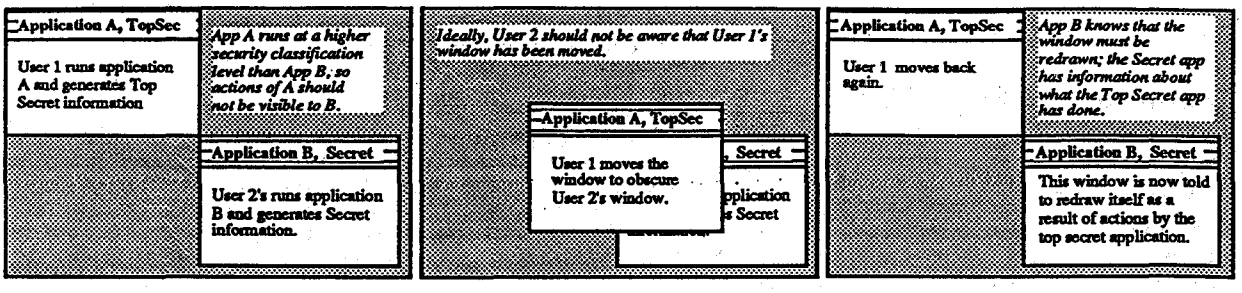


Figure 2: Pictorial Channel: Overlapping Windows

must have done the write into *Buffer* earlier. This sets up a subgoal that must be achieved before goal 3.

4. Current process = $User_2$

   Possible Plans for Goal 4:

   - *Swap*

   Choose plan *Swap* and propagate the goal 3.

$$\overbrace{\ldots\ Swap}^{User_2}\ \overbrace{Cut(i)\ Store(i,k)}^{User_1}$$

   A plan for goal 3 may now be be applied. Possible Plans for Goal 3:

   - *Paste*

   Choose plan *Paste(i)*.

$$\overbrace{\ldots\ Paste(i)\ Swap}^{User_2}\ \overbrace{Cut(i)\ Store(i,k)}^{User_1}$$

   This sets up the final goal:

5. $Var(User_2, i) = Mem(i,k)_0$

   This may be achieved directly by plan *Fetch(i, k)*.

Result:

$$\overbrace{Fetch(i,k)\ Paste(i)\ Swap}^{User_2}\ \overbrace{Cut(i)\ Store(i,k)}^{User_1}$$

Since this plan was achieved by working backwards from a goal, the actual plan that would be followed to pass information between users via a buffer is:

$$\overbrace{Store(i,k)\ Cut(i)}^{User_1}\ \overbrace{Swap\ Paste(i)\ Fetch(i,k)}^{User_2}$$

The previous example described an obvious example of flow that was easily discovered by SPLAN. In that example, information was directly transferred from one user to another. This section describes a more subtle example of information flow, in which information is transferred indirectly. Here, one user will observe one of two possible results, depending upon the actions of a second user. Figure 2 shows how the information flow occurs. The general idea is that one user briefly moves a high-level object over a lower-level object, and then returns it to its original position. The lower-level object is informed that it has been obscured and must redraw itself, giving the lower-level object information about the activities of the higher-level object. SPLAN uses the initial state and the two possible final states to develop two plans, where the result obtained by User B depends upon User A's actions. In this particular example, the information flow occurs when A choses to (or refrains from) repositioning a high security window over a low security window (Figure 2).

Initial state:

$$\begin{pmatrix} View(U_A, i) = TOPSECRET - VAL \\ Loc(U_A, i) = ((0,0),(10,10)) \\ View(U_B, j) = SECRET - VAL \\ Loc(U_B, j) = ((15,15),(25,25)) \\ Var(U_B, num) = 0 \end{pmatrix}$$

Final state (User A moves a window):

$$\begin{pmatrix} View(U_A, i) = TOPSECRET - VAL \\ Loc(U_A, i) = ((0,0),(10,10)) \\ View(U_B, j) = SECRET - VAL \\ Loc(U_B, j) = ((15,15),(25,25)) \\ Var(U_B, num) = 1 \end{pmatrix}$$

Final state (User A does not move a window):

$$\begin{pmatrix} View(U_A, i) = TOPSECRET \\ Loc(U_A, i) = ((0,0),(10,10)) \\ View(U_B, j) = SECRET - VAL \\ Loc(U_B, j) = ((15,15),(25,25)) \\ Var(U_B, num) = 0 \end{pmatrix}$$

Final plan:

$$\overbrace{Swap}^{User_B}$$

$$\overbrace{Move(View(A,j),Loc(View(B,j)),over)}^{User_A}$$

$$\overbrace{Move(View(A,j),notLoc(View(B,j)),over)}^{User_A};$$

$$\overbrace{Swap}^{User_A}$$

$$\overbrace{Redraw\text{TOPSECRET}}^{User_B}$$

If the users are in collusion through a Trojan Horse in a program belonging to user A, user B can interpret the REDRAW caused by user A's actions as one bit of a message. If the users are not in collusion, user B can still gain some information about A's activities.

5.3 Exclusive access to critical sections

The concurrent algorithms for which SPLAN is most appropriate all have certain common characteristics. The most important is that processes each contain a *critical section* of code. The purpose of mutual exclusion is to prohibit more than one process from executing this section of code at a time. Processes may manipulate their own local variables, or shared variables; in general, the critical section of code is used to read or modify a shared variable.

Producing a correct algorithm for mutual exclusion is nontrivial; several such incorrect algorithms have been published. SPLAN can detect the flaw in Hyman's 'simplified' version of Dekker's Algorithm for mutual exclusion involving two processes; Hyman's algorithm was published in [10].

If an operating system designer uses Hyman's algorithm to enforce mutually exclusive access to files, for example, access control violations might well result. Suppose that two users, with clearances of top secret and secret, have read access to the same file, classified as secret. Further suppose that the security policy of the system permits users to raise the classification of files (high water mark). If the developer relies upon the incorrect mutual exclusion algorithm, the code shown in Figure 4 will permit the user having the lower clearance to read the information supplied by the user having the higher clearance.

6 Discussion

vspace-0.05in The methodology described in this paper can be used in conjunction with flow analysis to identify those formal flow violations that are real. The methodology can be used before verification and after more conventional testing has uncovered the more obvious bugs. So far, it has been applied to detection of security flaws in small systems: in addition to interprocess flows in Millen's simple operating system. The methodology has been used to detect flow within a Low Water Mark system having partially ordered security levels. SPLAN is an improvement on STRIPS, which essentially does an exhaustive search, since SPLAN is guided by domain-specific plan heuristics. The organization of secure systems makes them especially amenable to this type of search.

There are, of course, limitations to SPLAN. If it terminates successfully without a plan, then we have 'verified' that the described flaw is not present. Although the system can detect certain forms of loops, termination is not guaranteed; thus, SPLAN cannot be relied upon as a verification system.

Current efforts to improve SPLAN are focused upon increasing the size of the software system it can handle, and the number and type of flaws it can detect. Since SPLAN's planning engine is ultimately based upon backtracking, increasing the number of possible states decreases SPLAN's speed dramatically. We are investigating the use of ***slicing***: the identification of a subprogram S of a program P such that S has the same functional behavior as P with respect to a property of interest. Similarly, a slice of a specification contains only those terms that bear upon a property. In effect, this will permit SPLAN to work at a coarser granularity without losing necessary details, which is much more efficient. Further, we are adding heuristics that should permit SPLAN to construct plans more rapidly (based upon the tiger team approach used elsewhere), and are investigating the usefulness of permitting user input to guide SPLAN's search as it executes.

References

[1] A. Barr and E. Feigenbaum, editors. *The Handbook of Artificial Intelligence.* HeurisTech Press, 1982.

[2] Department of Defense. Department of defense trusted computer system evaluation criteria. Technical report, 008-000-00461-7, 1985.

[3] J. Epstein and J. Picciotto. Trusting X: Issues in building trusted X Window systems–or–What's not trusted about X? *14th National Computer Security Conference*, 2:619–629, October 1991.

[4] R. Feiertag. A technique for proving specifications are multilevel secure. Technical Report CSL-109, Technical Report, SRI International, 1980.

[5] R. Fikes and N. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

```
==============================
User with Top Secret Clearance
==============================
1 ::flag[i] = true ;
2 :: while turn <> i  do
3 ::      while flag[j]  do
4 ::          skip ;
5 ::        enddo ;
6 ::      turn = i ;
7 ::   enddo ;
8a::<critical section begins>
8b:: Open Secret file for reading  F
8c:: Raise classification of  F
8d:: Close file
8e:: Open Top Secret  F for writing
8f:: Write Top Secret info  F
8g:: Close file
8h::<critical section ends>
9 ::flag[i] = false ;
```

```
==========================
User with Secret Clearance
==========================
1 ::flag[i] = true ;
2 :: while turn <> i  do
3 ::      while flag[j]  do
4 ::          skip ;
5 ::        enddo ;
6 ::      turn = i ;
7 ::   enddo ;
8a::<critical section begins>
8b:: Open Secret file  F
8c:: Read info from  F
8d:: Close file
8e::<critical section ends>
9 ::flag[i] = false ;
```

Figure 4: Access control violation.

```
: findPlan(6, In, Out, Plan), updateState(In, Result, Plan)?

In = state(symtab([]),
   symtab([[turn, 0], [inCS1, false], [inCS2, false],
          [flag0, false], [flag1, true]]),
   symtab([[[incorrectPc0, 1], 0], [[incorrectPc1, 1], 0]]))
Out = state(symtab([]),
    symtab([[turn, dontcare], [inCS1, true], [inCS2, true],
           [flag0, dontcare], [flag1, dontcare]]),
    symtab([[[incorrectPc0, 1], dontcare], [[incor-
rectPc1, 1], dontcare]]))
Result = state(symtab([]),
       symtab([[turn, 1], [inCS1, true], [inCS2, true],
              [flag0, true], [flag1, true]]),
       symtab([[[incorrectPc0, 1], 11], [[incorrectPc1, 1], 11]]))

Plan = [[[incorrectPc1, 1], 1],
        [[incorrectPc1, 1], 2],
        [[incorrectPc1, 1], 4],
        [[incorrectPc0, 1], 1],
        [[incorrectPc0, 1], 2],
        [[incorrectPc0, 1], 11],
        [[incorrectPc1, 1], 8],
        [[incorrectPc1, 1], 9],
        [[incorrectPc1, 1], 2],
        [[incorrectPc1, 1], 11]]
```

Figure 5: SPLAN's detection of the flaw in Hyman's Algorithm.

[6] L. J. Fraim. SCOMP: a solution to the multilevel security problem. *IEEE Computer*, 16(7):26–33, 1983.

[7] D. Frincke, M. Archer, and K. Levitt. A planning system for the intelligent testing of software. *Fifth Annual Knowledge-Based Software Assistant Conference*, Sept 24-28 1990.

[8] D. Frincke, M. Archer, and K. Levitt. CTPLAN: A planning-based approach to automatically detecting flaws in concurrent algorithms. *Sixth Annual Knowledge Based Software Engineering Conference*, Sept 1991.

[9] M. Gasser. *Building a Secure Computer System.* Van Nostrand Reinhold Company, 1988.

[10] H. Hyman. Comments on a problem in concurrent programming control. *Communications of the ACM*, 9(1):45, January 1966.

[11] C. E. McDowell and D. P. Helmbold. Debugging concurrent programs. *ACM Computing Surveys*, pages 593–623, December 1989.

[12] N. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Company, 1980.

[13] L. Osterweil, L. Fosdick, and R. Taylor. Error and anomaly diagnosis through dataflow analysis. *Proceedings of Summer School on Computer Program Testing*, pages 35–63, 1981.

[14] J. Peterson and A. Silberschatz. *Operating systems concepts.* Addison-Wesley Publishing Company, 1987.

[15] J. Rushby. EHDM specification and verification system: Implementation of formal semantics. Technical Report SRI Project 8096, report A002, Technical Report, SRI International, SRI International Menlo Park, CA 94025, 1989.

[16] R. Scheifler and J. Gettys. The X window system. *ACM Transactions On Graphics*, 5(2):79–109, 1986.

[17] R. Seviora. Knowledge-based program debugging systems. *IEEE Software*, pages 20–32, May 1987.

[18] R. Taylor. A general purpose algorithm for analyzing concurrent programs. *Communications of the ACM*, 26(5):362–376, May 1983.

[19] J. Tsai, K-Y Fang, and H-Y Chen. Debugger for concurrent programs. *Proceedings 13th Annual International Computer Software and Applications Conference*, September 1989.

[20] R. Wilensky. *Planning and Understanding.* Addison-Wesley Publishing Company, 1983.

A FOUNDATION FOR COVERT CHANNEL ANALYSIS[1]

Todd Fine
Secure Computing Corporation
1210 West County Road E, Suite 100
Arden Hills, Minnesota 55112
fine@sctc.com

Abstract

Two different definitions of "covert channels" are discussed, that used by information flow tools and that assumed by noninterference. A proof is given that any system that is secure with respect to flow tools is secure with respect to noninterference. Examples are provided to demonstrate the converse does not hold. We argue that noninterference provides a better definition of "covert channel" since the information flows identified by flow tools and not by noninterference are "formal flow violations". In addition, the practice of assuming tranquility when performing a covert channel analysis is questioned and an information flow tool policy for nontranquil systems is developed.

INTRODUCTION

In this paper, we contrast the definition of "covert channel" used by certain classes of information flow tools[3] with the definition of "covert channel" assumed in a *noninterference analysis*[1, 5, 2]. To do so, we state security policies corresponding to each definition of "covert channel" and compare the policies. We refer to the policies as the *ft-policy* and the *ni-policy*.

The original motivation for this work was to justify the use of noninterference to analyze LOCK^{TM}[6]. Since prior efforts to perform covert channel analysis relied on information flow tools, it was important to clearly understand the relation between a noninterference analysis and an information flow tool analysis.

The, perhaps, surprising conclusion is that the two policies are not equivalent; any system which has been shown to be secure with respect to the ft-policy satisfies the ni-policy, but there are some systems that satisfy the ni-policy while being insecure with respect to the ft-policy. Although this makes it appear that using an information flow tool provides a more complete analysis than using a noninterference policy, we argue that the converse is true because the information flows that are identified by an information flow tool but not by a noninterference analysis do not actually pose any threat.

Although there are similarities between the characterization of flow tools here and that in [5], there are two significant differences: 1) we do not assume tranquility, and 2) in addition to determining an ft-policy that implies the ni-policy, we consider whether the ni-policy implies the ft-policy.

The relevance of the first point is that some formal flow violations[2] are the direct result of tranquility being assumed in nontranquil systems. Since the ft-policy developed here applies to nontranquil systems, flow tools that previously required tranquility can be extended to address nontranquil systems by changing their policy to the policy developed here. In fact, modifications recently made to the Ina Flo tool [7] were motivated by the ft-policy developed here.

The relevance of the second point is that a better understanding of formal flow violations can be obtained by considering why the ft-policy is overly restrictive.

[1] This paper is based on work performed under contract MDA904-87-C-6011 with the U.S. Government, Maryland Procurement Office (MPO).

[2] A formal flow violation is an information flow that does not represent an illicit flow yet is identified as an illicit flow by an information flow tool. Formal flow violations are sometimes referred to as false positives.

As the ft-policy is a natural extension of the Bell-LaPadula Policy (BLP), these results suggest that a firmer foundation for covert channel analysis can be obtained by defining "covert channel" in terms of noninterference rather than attempting to extend the BLP paradigm.

BACKGROUND

MLS systems to date are motivated by practices in the paper and pencil world; each piece of data is assumed to be labeled with a sensitivity level and each user is assumed to have an assigned set of sensitivity levels to which he is cleared; a user is only permitted to observe data when he is cleared to the data's sensitivity level. Although this analogy is natural, it is flawed in that a critical assumption that is implicit in the paper and pencil world is invalid in the context of MLS systems. *While the individual causing an "information flow" in the paper and pencil world is aware of the "information flow", users of MLS systems are often unaware that "information flows" are occurring.*

For example, it is reasonable to expect an individual to be aware that an "information flow" occurs when he moves a page between two folders. Because the individual is aware of the "information flow", it is reasonable to expect him to ensure that the second folder is labeled appropriately for the sensitivity level of the page that has been moved. On the other hand, when an individual executes a program, he often has no idea what actions are occurring inside the system. Thus, it is not reasonable to expect the individual to ensure that whenever information is moved from one object to another, the sensitivity level of the target data item is appropriate for the transmitted information.

BLP addresses this by constraining the actions permitted by processes operating on a user's behalf. Each process is assigned an access level and it is required that: 1) each process may only observe objects having a sensitivity level dominated by its access level, 2) each process may only modify objects having a sensitivity level dominating its access level.

Suppose a process p causes information to flow from object obj_1 to object obj_2. Then, BLP requires that p's access level dominates obj_1's level and is dominated by obj_2's level. Consequently, the sensitivity of the target for the information flow dominates the sensitivity of the source for the information flow. This suggests that BLP is sufficient to prevent information flow downward in security level. In fact, it is typical to discover covert channels even in systems enforcing BLP. The problem is more in the manner in which BLP is applied than an inherent flaw in BLP.

Two "errors" are commonly made when applying BLP. First, the set of system objects is defined to be entities such as files and directories; entities such as kernel variables and hardware registers are ignored. Obviously, any covert channels through the ignored entities cannot be discovered by analysis with respect to this formulation of BLP. Second, it is assumed that there is an access matrix that is consulted whenever objects are observed or modified. For example, rather than requiring that a process can only observe lower level objects, it is required that whenever the access matrix indicates a process can observe an object, the object is at a lower level. The separate issue of ensuring that the access matrix is always consulted before allowing an object to be accessed is not addressed by BLP.

Information flow tools such as the Gypsy Information Flow Tool (GIFT)[3][3] attempt to extend BLP to address both of these deficiencies, but are still inadequate. First, they are typically conservative in their analysis in that they often identify formal flow violations. Second, they provide no support for distinguishing between covert channels and formal flow violations. In the following, we examine the cause of formal flow violations and illustrate how they can be avoided using noninterference.

[3]Since we are more familiar with the GIFT than any other information flow tool, our ft-policy is greatly influenced by the GIFT. Although we have attempted to obtain a fairly general result, the degree to which our results apply to other flow tools is not yet clear.

DEFINITIONS

In this section, we define a simple system model and use it to state the ft-policy and the ni-policy. We use a state machine model of the system with $\mathcal{ST}$ denoting the set of system states, $\mathcal{OPS}$ denoting the set of system operations, and st^x denoting the state resulting from applying x to st, where x is either a single operation or a sequence of operations.

To state the policies, we assume a set of levels, $\mathcal{L}$, that is partially ordered by $\preceq$.

FT-POLICY

To state the ft-policy, we assume that a state is a mapping from state components to values. Formally: $\mathcal{V}$ is a set of values, $\mathcal{C}$ is the set of state components, and a state is a mapping from $\mathcal{C}$ to $\mathcal{V}$. We use $st(c)$ to denote the contents of c in st.

The ft-policy requires that a security level be assigned to each state component. If levels are assigned statically, the system is tranquil; otherwise, it is nontranquil. The disadvantage of a static level assignment is that it can lead to formal flow violations.

For example, consider the hardware registers available to processes executing in the system. Since these registers can be both read and written by the current process, they must have the same level as the current process. Thus, it is not possible to statically assign a level without introducing formal flow violations.

Since each channel identified must be analyzed, a great deal of extra work might need to be done as the result of identifying formal flow violations. When the set of objects is meant to model all state components rather than only the files and directories, the system is usually nontranquil. The sensitivity levels of components such as the hardware registers and kernel variables change as processing proceeds. Consequently, we assume a dynamic level assignment with $level(c, st)$ denoting the level of component c in state st.

Information flow tools work by identifying *targets* and *sources* for information flow. A target for a system operation is a state component that is changed when the operation is executed. We say that a state component is changed when either its value or its level is changed. The set of sources for a target is the set of state components which determine the modifications made to the target. Note that the sources and targets may depend upon the state of the system when the operation is executed. For example, a request to write data to a file has the specified file as target only when the request is executed from a state in which the accesses for the file indicate that it may be written.

To clarify the definitions of sources and targets we formalize them as follows:

- $targets(i, st)$ is the set of components whose value or level is altered when i is executed in st
- $sources(t,i,st)$ is the set of sources for the information flow into t when i is executed in st; rather than defining *sources* we simply assume that whenever t is a target for i in st and all of the sources have the same value in st and st_1, then:

  $$t \in targets(i, st_1) \text{ and } (st^i)(t) = (st_1{}^i)(t) \text{ and } level(t, st^i) = level(t, st_1{}^i)$$

  In other words, whenever all of the sources for an operation have the same values in two states, the targets for the operation are the same in both states and the targets are changed in the same manner in both states. If this condition were not satisfied, then the modifications made by an operation would be dependent on information other than the sources.

Note that we have not actually defined *sources*; instead we have only placed a requirement on its definition. Given a target for an operation, flow tools use a set of rules to identify a set of state components that satisfies the characterization of *sources* stated above. Since these rules vary from tool to tool, different flow tools might generate different sources for the same information flow. By limiting the assumptions we make about *sources*, we make our work more generally applicable.

Using these definitions it is possible to state the ft-policy. It simply requires that:

$$t \in targets(i, st) \text{ and } sc \in sources(t,i,st) \Rightarrow (level(sc, st) \preceq level(t, st^i) \text{ and } level(sc, st) \preceq level(t, st))$$

The first requirement is the obvious requirement that high-level source data from st not be used to compute the value stored in a low-level state component in st^i. The second requirement is less obvious. It requires that a target's level in st dominate the levels of each of its sources in st. This prohibits a high-level subject from causing a low-level object to be reclassified at a high-level. Without this requirement, the policy would not prohibit a high-level subject from transmitting information downward in level by altering the set of objects visible at the low-level. Since the two requirements are identical in tranquil systems, flow tools that assume tranquility only need to generate the first requirement. By doing so, they might fail to address certain covert channels if the system being analyzed is actually nontranquil. Figure 1 illustrates the two types of threats addressed by the ft-policy.

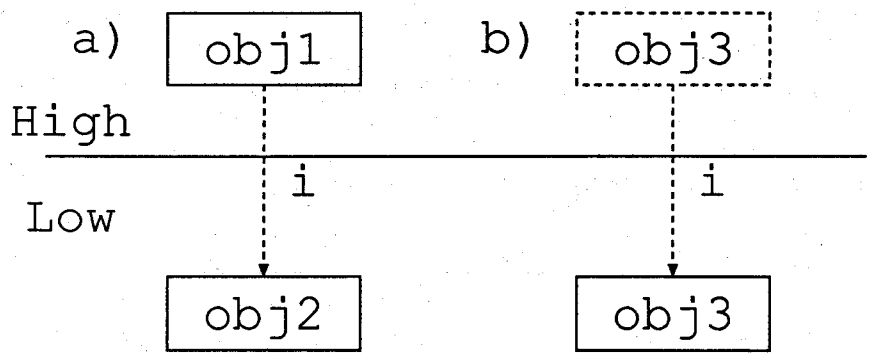


Figure 1: a) i has a low-level object as a target and a high-level object as a source, b) i reclassifies a high-level object as a low-level object

Note that this policy attempts to address both of the "errors" commonly made when applying BLP. First, it assigns sensitivity levels to all state components rather than only assigning levels to files and directories. Second, it addresses all information flows rather than only requiring that the accesses permitted by the access matrix are consistent with the assigned levels.

NI-POLICY

To state the ni-policy, we assume:

- There exists an equivalence relation on states, $st_1 \approx_l st_2$, capturing when st_1 and st_2 "look the same" to subjects at level l. Intuitively, $st_1 \approx_l st_2$ holds when the data visible to subjects at level l is the same in both states.
- Each operation is executed by a subject at some security level.

We define $seq|l$ to be the sequence obtained from seq by removing all operations executed by subjects at levels not dominated by l. In other words, $seq|l$ is the portion of seq visible to subjects at level l.

Now, the ni-policy can be stated as:

$$st_1 \approx_l st_2 \Rightarrow st_1{}^{seq} \approx_l st_2{}^{seq|l}$$

This is a very natural requirement; given that st_1 and st_2 appear the same to subjects at level l and that seq and $seq|l$ appear the same to subjects at level l, $st_1{}^{seq}$ and $st_2{}^{seq|l}$ should appear the same to subjects at level l. This is illustrated in Figure 2.

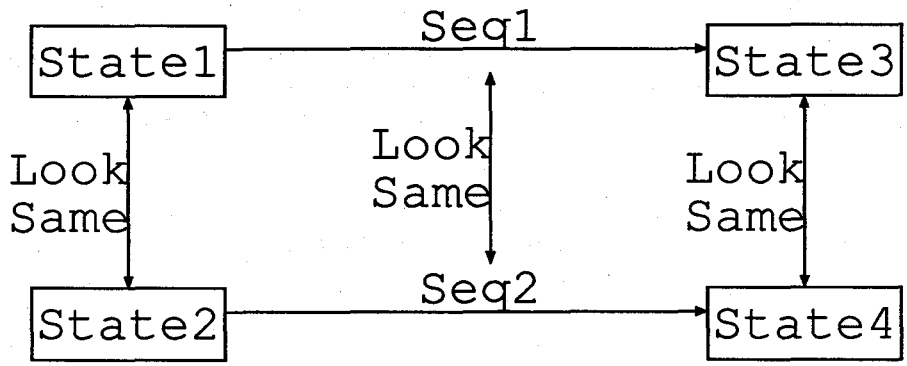


Figure 2: The ni-policy. If st_1 and st_2 look the same to subjects at level l and seq_1 and seq_2 look the same to subjects at level l, then the resulting states look the same to subjects at level l.

Now that we have defined the policies, we consider the relationship between them.

COMPARISON OF POLICIES

In this section we show that 1) Any system that is ft-secure is ni-secure[4], and 2) The converse does not hold.

In obtaining this result, we assume that given any target for an operation, the associated set of sources contains at least one component at the level at which the operation is executed. This assumption, which we refer to as **Op Assumption**, says that if an operation is executed at level l_i, then, regardless of the state in which the operation is executed, there is some state component at level l_i that influences the changes made to every target. In most systems, each operation is associated with a client subject and the level at which the operation is executed is the level of the client subject. So, this assumption is satisfied by requiring that the client subject is a source object for every target. It is not unusual for a flow tool to make this assumption.

It is also important to note that it is necessary to determine the level at which each object can be accessed to use either policy. For ft-policies, this is done by assigning a sensitivity level to each state component; for ni-policies, this is done by defining $\approx_l$. Other than the obvious requirement that outputs visible at a level l must be labeled with a level greater than or equal to l, the analyst is free to assign sensitivity levels or define $\approx_l$ as he sees fit. The system is ft-secure if there is at least one level assignment such that the ft-policy is satisfied and is ni-secure if there is at least one definition of $\approx_l$ such that the ni-policy is satisfied.

NI DOES NOT IMPLY FT

[4] We use ft-secure and ni-secure to mean secure with respect to the ft-policy and ni-policy, respectively.

In this section we demonstrate that it is possible for a system to satisfy an ni-policy even though it does not satisfy the corresponding ft-policy. There are essentially two reasons. First, it is not always possible to define a level assignment function in a natural way from a given definition of $\approx_l$. Second, ft-policies are not flexible enough to take into account dependencies between state components. We now provide examples for each of these concerns.

Consider the following system. The system has three integer-valued state components. We will denote the values of these state components in a given state of the system st by $st.X$, $st.h1out$, $st.h2out$. The system has four security levels, $high_1$, $high_2$, low_1, and low_2. $High_1$ and $high_2$ dominate low_1 and low_2; $high_1$ and $high_2$ are incomparable; and low_1 and low_2 are incomparable.

$St.h1out$ is a data buffer at level $high_1$ for processes at level $high_1$. Similarly, $st.h2out$ is a data buffer at level $high_2$ for processes at level $high_2$.

Processes with level low_1 or low_2 can invoke only the *Write* operation. This operation sets $st.X$ to v, a parameter specified in the operation. Processes with level $high_1$ or $high_2$ can invoke only the *Read operation.* This operation copies the value of $st.X$ to either $st.h1out$ or $st.h2out$ depending on the level of the client subject.

Consider what level to assign to $st.X$ to demonstrate the system is ft-secure. Since $st.X$ can be modified from both low_1 and low_2, its level must dominate both low_1 and low_2. This means that its level must be $high_1$ or $high_2$. Since $st.X$ can be observed from both $high_1$ and $high_2$, its level must be dominated by both $high_1$ and $high_2$. This means that its level must be low_1 or low_2. So, there is no level that we can assign to $st.X$ that will result in the system satisfying the ft-policy. This demonstrates the first problem with an ft-policy. There are times when there is no level that can be assigned to a state component. This is illustrated in Figure 3.

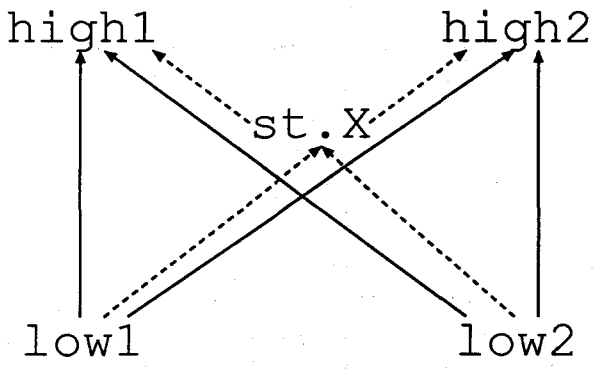


Figure 3: A state component for which there exists no meaningful security level. The solid lines indicate the ordering imposed on the levels. The dashed lines indicate the relations that must hold between the level of $st.X$ and the other levels for the system to be ft-secure. Obviously, the level that must be assigned to $st.X$ does not correspond to any of the levels in the system.

To see that this in not a problem when using an ni-policy, 1) define all states to be equivalent with respect to low_1 and low_2, and 2) define two states to be equivalent with respect to $high_1$ and $high_2$ if and only if the X component and the out buffer corresponding to the level have the same value in both states.

Now, if ops is a sequence of operations, $ops|high_1$ is obtained by removing any *Read* operations executed from $high_2$. So, the only difference between st^{ops} and $st^{ops|high_1}$ is in the $.h2out$ component and $st^{ops} \approx_{high_1} st^{ops|high_1}$. Similar reasoning shows that $st^{ops} \approx_{high_2} st^{ops|high_2}$. Since all states are equivalent with respect to either of the low levels, it is clear that the ni-policy holds even though the ft-policy does not.

The problem here is that the set of levels at which $st.X$ is visible is $\{high_1, high_2\}$. Since this set has no greatest lower bound, there is no way to correctly assign a level to $st.X$. Although it might be possible to address this problem by extending the set of security levels to a lattice, we will not consider this possibility here because there would still be a more serious problem.

Consider a system having four integer-valued state components, $st.A$, $st.B$, $st.lout$, and $st.hout$ and two security levels, *high* and *low*. $St.lout$ and $st.hout$ are data buffers for, respectively, low-level and high-level processes and thus have, respectively, levels *low* and *high*.

The only operations available to low-level processes are *Low Read* and *Low Write*. *Low Read* copies the value of $st.A - st.B$ to $st.lout$, while *Low Write* performs the assignment $st.A \leftarrow st.B + v$, where v is a value specified by the client. In effect, these operations allow low level processes to store values in the psuedo state component $st.A - st.B$.

The only operations available to high-level processes are *High Read* and *High Write*. *High Read* copies the value of $st.B$ to $st.hout$, while *High Write* atomically performs the assignments $st.A \leftarrow st.A - st.B + v$ and $st.B \leftarrow v$, where v is a value specified by the client. These operations allow high level processes to store values in $st.B$ while not altering the value low level processes have stored in $st.A - st.B$.

The *Low Read* operation has $st.lout$ as a target and $st.A$ and $st.B$ as sources (since both $st.A$ and $st.B$ influence the value written to $st.lout$). In order for the flow from $st.A$ and $st.B$ to $st.lout$ to be secure, $st.A$ and $st.B$ must have a security level of *low*. The *High Write* operation has $st.A$ and $st.B$ as targets and the high-level client process as the source. In order for the flow from the client process to $st.A$ and $st.B$ to be secure, $st.A$ and $st.B$ must have a security level of *high*. So, regardless of the manner in which $st.A$ and $st.B$ are assigned levels, either *Low Read* or *High Write* has an insecure information flow. Consequently, the system cannot satisfy an ft-policy.

To show that the system satisfies an ni-policy, we define 1) two states to be equivalent with respect to *low* if the low level output buffer and the difference between the A and B components are the same in both states, and 2) two states to be equivalent with respect to *high* if the high level output buffer and the B component are the same in both states.

Note that *High Read* cannot alter $st.A$, $st.B$, or $st.lout$. So, if st' is the state obtained by executing a *High Read* operation in st, then st and st' are equivalent with respect to *low*. Although, *High Write* can alter $st.A$ and $st.B$, it cannot alter their difference. So, a similar result holds for *High Write*.

Now, suppose st'_1 and st'_2 are the states resulting from applying an operation to st_1 and st_2 and that $st_1 \approx_{low} st_2$. Then, the definitions of *Low Read* and *Low Write* are such that $st'_1 \approx_{low} st'_2$.

From these observations, it is quite easy to demonstrate that the ni-policy holds. High-level operations cannot change the view at level *low* while low-level operations do not make use of information that is not visible at level *low*.

This example shows that it is possible for a system to satisfy an ni-policy even though it does not satisfy the corresponding ft-policy. The problem with the ft-policy is that it cannot recognize that even though both high and low level processes can observe and modify $st.A$ and $st.B$, the operations in the system prevent them from doing so in a manner that would allow information to flow to the other level.

FT IMPLIES NI

Although systems can satisfy the ni-policy without satisfying the ft-policy, any system that is ft-secure is ni-secure. This is captured in the following theorem (a more formal statement and proof can be found in [4]):

- Theorem: If a system satisfies the ft-policy with a particular level assignment function and **Op Assumption** holds, then there exists an equivalence relation $\approx_l$ for which the system satisfies the ni-policy.
- Proof Sketch: We must show that $\forall seq, st_1, st_2, l : st_1 \approx_l st_2 \Rightarrow st_1{}^{seq} \approx_l st_2{}^{seq|l}$.
  - Define $st_1 \approx_l st_2$ as $\forall c$, $st_1(c) \approx_l st_2(c)$, where $st_1(c) \approx_l st_2(c)$ means:

    $level(c, st_1) \preceq l \Leftrightarrow level(c, st_2) \preceq l$

    and if $level(c, st_1) \preceq l$ and $level(c, st_2) \preceq l$, then $st_1(c) = st_2(c)$
  - Suppose t is a target of an operation. Since the system is ft-secure, both the old and new level of the target must dominate the level of any of the sources. The **Op Assumption** requires at least one of the sources to be at the level at which the operation is executed. So, the old and new level of any target dominate the level at which the operation is executed.
  - Suppose $st(c) \not\approx_l st^i(c)$. Then, c is a target of i since either its level or value is altered by i. Thus, its level must dominate that of l_i, the level at which the operation is executed, in both st and st^i. If l_i is not dominated by l, then the transitivity of the dominates relation implies that the level of c must not be dominated by l in either state. This would be a contradiction since $st(c) \approx_l st^i(c)$ holds whenever c's level is not dominated by l in either state. Consequently, the only operations that can alter entities visible at or below level l are those operations executed at levels dominated by l.
  - Now, suppose $st_1 \approx_l st_2$ and consider an arbitrary component c. If c's level is not dominated by l in either $st_1{}^i$ or $st_2{}^i$, then $st_1{}^i(c) \approx_l st_2{}^i(c)$. Otherwise, the ft-policy requires that the levels of all of the sources be dominated by l in either st_1 or st_2. Then, the definition of *sources* requires that c have the same value and level in $st_1{}^i$ and $st_2{}^i$. So, in either case, $st_1{}^i(c) \approx_l st_1{}^i(c)$.
  - This analysis shows that whenever two states look the same at level l, the states resulting from applying any operation look the same at level l. Combining this with the observation that operations executed at levels that are not dominated by l do not alter anything visible at level l it is clear that the state resulting from applying a sequence seq of operations looks the same at level l as the state resulting from applying $seq|l$. Thus, the ft-policy and the **Op Assumption** are sufficient to establish the ni-policy.

This shows that any system that is demonstrated to be secure using a flow tool can be demonstrated to be secure using a noninterference policy.

CONCLUSION

Without a generally accepted definition of what a covert channel is, it is not possible to prove either of the definitions proposed in this paper correct. However, based on the examples presented in this paper, it is reasonable to conjecture that the definition in terms of interferences between subjects is a better definition. The examples of systems that satisfy the ni-policy even though they do not satisfy the ft-policy seem intuitively secure. If these systems are accepted as being secure, then the definition of covert channels in terms of flows between sources and targets must be accepted as being too restrictive. As the ft-policy is a natural extension of BLP, the analysis in the preceding sections also suggests that BLP should not be viewed as the overall system policy.

Although BLP and the ft-policy have deficiencies, this does not mean that they cannot be useful. Analysis with respect to BLP can often provide support for an analysis with respect to the ni-policy. Furthermore, tools have been constructed to simplify analysis with respect to an ft-policy, while no tools have yet been

constructed to simplify an analysis with respect to an ni-policy. Thus, analysis with respect to an ft-policy is typically more automated than analysis with respect to an ni-policy.

Still, it is important to understand the limitations of these approaches. If one accepts that the definition of covert channels in terms of flows from sources to targets is too restrictive, then one must accept that tools based on the ft-policy are forever doomed to be overly conservative. No matter how much additional work is spent on such tools, there is always the possibility of formal flows being identified. This suggests that a trade-off must be made between the manual effort required to distinguish between formal flow violations and covert channels and the manual effort required to perform analysis with respect to the ni-policy.

An important area for future research is comparing the policy enforced by flow tools other than the GIFT to the ni-policy. If there are flow tools that enforce policies that are identical to the ni-policy or closer to the ni-policy than the ft-policy stated here, then those flow tools might be more useful than tools enforcing the ft-policy. In any case, it would be interesting to know the relationship between the policies enforced by the various flow tools and the ni-policy.

Acknowledgments

Thanks to J. Thomas Haigh and Richard O'Brien for reviewing this paper.

References

[1] Todd G. Fine, J. Thomas Haigh, Richard C. O'Brien, and Dana L. Toups, Noninterference and Unwinding for LOCK, *Proceedings of the Computer Security Foundations Workshop II*, 1989, pp. 22-28.

[2] J. Thomas Haigh and William D. Young, Extending the Noninterference Version of MLS for SAT, *IEEE Transactions on Software Engineering*, Volume 13, February 1987, pages 141-150.

[3] John McHugh, Robert L. Akers, and Millard C. Taylor, *GVE Users Manual: The Gypsy Information Flow Tool, A Covert Channel Analysis Tool*, Computational Logic, Incorporated, 1989.

[4] Todd Fine and Barry Miracle, *LOCK Covert Channel Analysis*, Secure Computing Technology Corporation, 1990.

[5] John Rushby, *Mathematical Foundations of the MLS Tool for Revised Special*, SRI International, 1984.

[6] O.S. Saydjari, J.M. Beckman, and J.R. Leaman, LOCK Trak: Navigating Uncharted Space, *Proceedings of the 1989 IEEE Computer Society Symposium on Security and Privacy*, 1989, pp. 167-175.

[7] Steven T. Eckmann, *An Information Flow Model for FDM (Draft)*, Unisys Defense Systems Inc., 1991.

GENERAL ISSUES TO BE RESOLVED IN ACHIEVING MULTILEVEL SECURITY (MLS)

Bill Neugent
The MITRE Corporation
7525 Colshire Dr.
McLean, VA 22102, U.S.A.
703-883-6632

1. Introduction*

Lack of Multilevel Security (MLS) within United States (US) Department of Defense (DOD) computer systems is recognized as a significant shortcoming, because it limits interoperability and data fusion. To help address this problem, the Joint MLS Technology Insertion Program was officially established in January 1990. The program is managed by the Defense Information Systems Agency (DISA) and the security coordinator is the National Security Agency (NSA). The purpose of the program is to expedite the fielding of MLS operational capabilities within DOD. This paper is derived from guidance produced by the program [1].

This paper summarizes the issues that underlie and drive efforts to achieve MLS, along with a proposed strategy in each area. The Joint MLS Technology Insertion Program by itself does not have the authority or resources to resolve all of these issues, but it can help to identify the issues and marshal resources to address them. Table 1 provides a summary.

2. High User Expectations

Users desire affordable, easily-implemented operational enhancements that can be implemented within a year or two. Especially with the number of trusted products now coming available, users will be vulnerable to exaggerated or overly optimistic claims about the new products.

The strategy is to field security guards or other limited commercial solutions immediately, while planning for subsequent evolution. The strategy also is to support and encourage a conservative, realistic view of trusted products. For example, when users state a requirement for MLS, what they often envision is a system that can support unclassified through TOP SECRET data at no extra cost and with no loss of functionality or performance. Such users must be educated in the vision of MLS as it might realistically be achieved, rather than MLS as it is idealistically visualized.

---

* This paper is based on work performed under Contract DAAB07-91-C-N751 for the Defense Information Systems Agency (DISA).

Table 1. Issues and Strategies

Issue	Strategy
Users desire quick, affordable, simple solutions	Field limited solutions where appropriate; encourage realistic expectations
B2 products are needed; most are B1	Use B2 products if feasible; use B1 and B1+ products with operational restrictions
Integration of diverse products is difficult	Keep initial efforts simple; use testbeds; develop system-wide security policy
Current trusted products are incomplete	Use where feasible; identify needed capabilities
Certification is complex and requires scarce skills and substantial time	Work to simplify process; ensure supported MLS certifications are adequate
Accreditors might approve systems for MLS without adequate safeguards	Encourage compliance with policy; ensure adequate certification resources
Trusted products often are too difficult to manage and use	Review early for these qualities; balance operational and security needs
Critical guidelines and standards still are evolving	Work closely with such efforts; ensure integration and completeness
DOD funding cutbacks threaten some efforts	Emphasize return on investment, especially on near-term efforts

3. Effective Use of B1 Products

A critical issue is the effective use of B1 products, which are those products designed to satisfy the requirements for a class B1 system in accordance with the Trusted Computer System Evaluation Criteria (TCSEC). The issue with B1 products has two aspects. First, B1 products are much easier to achieve technologically than B2 products and as a result are the primary targets of vendor efforts, especially for workstations and Database Management Systems (DBMSs). Second, according to enclosure 4 of DOD Directive 5200.28, B1 products can be trusted only in environments where the risk range is one [2]. An example of an environment with a risk range of one is one in which a system contains SECRET data and supports some users cleared only to CONFIDENTIAL. Almost all DOD environments requiring MLS capabilities have a greater risk range than one. The bottom line is that products are of little use unless they can support a risk range of at least two, which would be sufficient to separate TOP SECRET data from SECRET-cleared users -- a key requirement. According to the DOD Directive, B2 products can support a risk range of two, but B1 products cannot.

So in the near term and beyond, the strategy is to use B2 and above products where they are available and meet user needs and to encourage further development of B2 products. B2 products are the preferred near-term targets, because they should be attainable in the next few years and because there are legitimate security reasons for the B2 assurance requirements (e.g., system architecture, configuration management).

Nevertheless, while it would be desirable to focus primarily on B2 products, currently there are more B1 than B2 or above products. Furthermore, some B1 products meet critical user requirements that B2 products do not meet, such as the workstation requirement for trusted window management. So the strategy is also to experiment with

B1 products in the near term and to use them operationally, but only with acceptable operational restrictions. Although B1 security is not ideal, B1 products still permit an investigation of MLS interoperability issues and a determination of what functionality is needed in MLS systems.

Operational restrictions are needed with B1 products, however. An example of an operational restriction that might be appropriate in some cases is to require all system users to be cleared to the highest level of data supported by the system, but to allow users to access remote systems that operate at lower security levels. Note that a threat analysis can be helpful in identifying which restrictions best counter threats and could reveal that a particular threat environment does not warrant B2 protection (or, in the opposite case, warrants greater protection). B1 products such as Compartmented Mode Workstations (CMWs) that include useful B2 and B3 features and assurances are preferred over products that are only minimally B1; such B1+ products would require fewer operational restrictions.

The case might be made that, in using B1 products where B2 or higher products are preferred, DOD undermines the market for B2 or higher products. On the other hand, DOD has actively supported the many efforts to develop B1 products by working with the vendors and evaluating the products. For DOD now to find little operational use for such products might undermine the market for trusted products in general, including B2 and above products. The strategy thus is to take a balanced approach, using available products in the near term and fully exploiting B2 and above products as they come available.

Continued emphasis on B3 or higher products also is important. These products, though involving greater development risk, have a greater potential payoff. The higher development risk derives from the technical difficulties in developing B3 or higher products. The greater potential payoff is due to the increased trustworthiness of the products and the increased range of security levels supportable. Aside from the greater potential payoff, another reason to emphasize B3 or higher products is to ensure a marketplace for such products.

4. Integration

To date, the TCSEC and related guidance have focused on particular types of products; little attention has been placed on integrating different products. It has become clear, however, that careful integration is necessary for effective MLS operation. For example, it cannot be assumed that the combination of two trusted products is trusted. Integration risks exist when integrating:

- Multiple homogeneous components that were designed for standalone security operation
- Multiple components from different vendors
- Heterogeneous components, e.g., workstations, hosts, DBMSs, guards, and network products
- Products built to different levels of assurance, e.g., a commercial biometric authentication capability and a B2 workstation

- o MLS and system high components
- o Nonsecure commercial applications and trusted commercial components
- o Trusted products and operational field applications
- o Products that enforce different security policies

The integration risks are that the sum of the products (1) might not work correctly, (2) might not provide complete or correct enforcement of security policies, (3) might invalidate the security of individual component products, and (4) might introduce new security problems outside the scope of any single component.

The strategy to address this issue is to use testbeds to integrate limited, initial configurations and to identify and encourage the development of the missing pieces. Furthermore, a system-wide security policy must be prepared to ensure that the multiple products involved work together correctly.

5. Limited Capabilities of Current Products

Closely related to the issues of using B1 products and integrating different products is the fact that current products are quite limited in their capabilities. For example, some user interfaces to trusted DBMSs are not yet MLS, necessary trusted communication protocol software does not yet exist. In general, vendors are aware of these shortcomings and trusted product capabilities will improve as the technology and marketplace mature.

The strategy is to expedite evolution of both the technology and the marketplace by using current products to the extent feasible and by identifying, encouraging, and if necessary supporting development of needed capabilities and changes. An initial list of significant needed capabilities is as follows:

- o Trusted communication protocol software, e.g., Transmission Control Protocol (TCP)/Internet Protocol (IP)
- o Security labeling standards that permit integrated labeling among operating systems, DBMSs, network protocols, and selected applications (e.g., messages)
- o B2 workstations with acceptable user interface, capability, and performance
- o B2 DBMSs with acceptable user interface, capability, and performance
- o Commercial Communications Security (COMSEC) Endorsement Program (CCEP) and Secure Data Network System (SDNS) products that are trusted both for COMSEC and Computer Security (COMPUSEC)
- o Trusted e-mail
- o Trusted central security management of distributed trusted workstations and servers, including central auditing
- o Strengthened authentication

- o Simplified security management interface
- o Strengthened audit data analysis, reduction, and archiving
- o Trusted applications

6. Certification

Another issue involves the certification of MLS systems. Certification is the technical assessment of whether a system meets its security requirements [3]. The major danger in certification is that the technical assessment of security will not be adequate (e,g., substantial vulnerabilities will be overlooked), due to lack of sufficient certification resources or properly qualified certification personnel. This is a fundamental problem.

An important aspect of the problem is certification complexity. For example, Evaluated Products List (EPL) operating system evaluations are narrow in scope; while they address TCSEC compliance, they do **not** address trusted network interfaces. Yet most EPL-rated operating systems have little operational utility unless they support network communication. The certification issue is that, when such a product is used in a network, the rating of the product cannot be assumed to apply to the combined product and network. Because of this, the certification done to assess the combined product and network must reassess areas that were addressed in the NSA product evaluation. For example, the Verdix Secure LAN (VSLAN) EPL summary states that combining VSLAN with other trusted components such as MLS hosts "may introduce new covert channels or penetration scenarios that were not evident from the evaluation of either component by itself; a complete network system must always be evaluated as a whole to ensure that the components together enforce the overall policy" [4]. This need for additional evaluation can add substantial complexity to the certification effort, especially if the system includes not only workstations and networks, but also DBMSs and trusted applications.

Yet despite the narrow scope of the NSA product evaluation, such an evaluation might take a calendar year or two (in part, because it is done in parallel with product development). This often is longer than the lifespan of the particular software version being evaluated. Certification reviews, being more broad in scope than product evaluations, introduce additional complexities, yet typically must be done in less calendar time. As examples of certification complexities, certification procedures must accommodate (1) assessment of compliance with complex security requirements (including integrity and denial of service), (2) modification of evaluated products, (3) use of evaluated products in ways not encompassed by the evaluation, and (4) agreements across accreditation boundaries. Such complexities can make it difficult or impossible to find adequate time and resources for certification.

Certification for MLS requires the services of specialized experts in the particular technologies employed and must include penetration testing. The scope of certification efforts must encompass the entire integrated system, rather than be limited to a subset of the components involved [5]. Guidance and training are needed in certification, but the fact remains that the most important aspect of certification is the use of objective, qualified specialists to perform the work. Without this expertise, certification reports have a high likelihood of containing incorrect or misleading information.

The strategy in the near term is to ensure that certification plans and resources are adequate for the systems under scrutiny. For the longer term, the proposed strategy is to look for ways to reduce resources needed for certification and to support government efforts to produce certification guidance.

7. Accreditation

Accreditation is the management decision to operate the system [3]. The accreditation decision is based upon certification findings and other inputs. The major risk in accreditation is that accreditors might decide to operate systems without adequate security safeguards. For example, according to policy, CMWs and B1 workstations are to be used in situations where risks are minimal (e.g., CMWs are intended for use in compartmented mode, in which all users are cleared to the highest level of the data processed). However, the functional characteristics of these workstations are such that they can be used with no changes to support full-MLS operation (e.g., with uncleared users and TOP SECRET data). DOD policy strongly recommends against such usage, but accreditors have the authority to use trusted technology however they see fit [2]. Furthermore, some vendors, when questioned about how their trusted products can be applied, say only that the decision rests with the accreditor.

This situation could lead to fundamental changes in the meaning of MLS. That is, since many trusted products currently are targeting B1, some near-term MLS systems will be based on B1 technology. Since most meaningful MLS environments require at least B2 (see section 2), it seems inevitable that some accreditors will accredit B1 technology to suffice where B2 technology is desired. Ultimately, a body of accreditation precedents could exist that threatens to override current policy recommendations [2].

So this accreditation risk threatens not only the security of individual systems, which might be accredited to operate without adequate safeguards, but also the underlying policy infrastructure that defines what trusted technology is and how it should be used. Of course, the opposing risk also must be kept in mind -- that overly conservative accreditation decisions will result in the lack of needed operational capabilities. The purpose of MLS is not to maximize security, but to improve operational capabilities while maintaining sufficient security. Conceivably a decrease in security might be acceptable if there is a large gain in operational capabilities. So accreditation decisions must avoid both extremes.

This is an important issue area. The strategy is to provide accreditors with pragmatic, responsible guidance to follow in making their accreditation decisions. In general, the guidance is to ensure that accreditations comply with DOD policy and adequately address environmental needs, with full consideration of asset value, threats, vulnerabilities, and residual risks. This ensures that MLS accreditors are informed of policy and have adequate planning and resources for certification. Meanwhile, DOD has identified uniform accreditation policy as a critical area needing attention.

8. Ease of Management and Use of MLS Systems

Ease of management and use of MLS systems is a critical topic, because to date too much attention has been placed on making trusted products secure from the inside out (i.e., from an internal technical standpoint). Not enough attention has been placed on making trusted

products easy to manage and use. Unless trusted products are easy to manage and use, they either will be used ineffectively or will not be used at all.

With respect to ease of management, insufficient attention has been placed on managing (administrating) MLS systems. Some common commercial products (e.g., UNIX) are so complex to administer that system administrators must be highly trained and even then are susceptible to errors that deny service or cause serious security violations [6]. Since system administrator error is an important source of security violations and since most DOD organizations do not have additional personnel to dedicate to this role, security products must minimize the number of people and the amount of training required for security administration and must include adequate and understandable system documentation.

To reduce security management risks, tools are needed to simplify the task and guidance is needed in setting and managing initialization parameters and other security-related tables. Tools are needed to help administrators manage multiple systems (e.g., centrally change permissions and gather audit data). Tools are needed to analyze audit data. Management tools are evolving, but much progress is needed.

With respect to ease of use, many MLS development failures have resulted because products were unacceptably cumbersome or because they simply did not solve the user's MLS problem. An example of an unacceptably cumbersome product is one that forces users to log off and then log on to enter data at different security levels. Regarding not solving the user's MLS problem, there are many products that, taken by themselves, do not fully satisfy MLS needs. For example, an MLS operating system alone might not satisfy needs for MLS databases or e-mail. An MLS operating system alone cannot automatically sanitize data; application-unique trusted software is needed. Data received from a system high system might have to be downgraded by human review, even in an MLS system.

Other risks related to use of trusted technology include possible higher cost and reduced performance of security products, compared with products that do not provide sufficient security for MLS operation. While these impacts are steadily lessening, where they remain they can present major obstacles to MLS operation. An Armed Forces Communications and Electronics Association (AFCEA) Information Systems Security (INFOSEC) study concluded that "transparent INFOSEC is a critical system goal," in that INFOSEC must be user friendly and minimize introduction of performance degradation [7].

The strategy in these areas is to ensure that MLS efforts adequately address security management and satisfy user operational needs by incorporating reviews early in development and acquisition efforts to assess those topics.

9. Evolving Guidelines and Standards

Guidelines and standards that impact MLS are still evolving and will not be completed for several years. Much work is needed, especially in security labeling, where there is a need for label compatibility among communication protocols, DBMSs, operating systems, and applications. Such label compatibility is needed both for product integration and for interoperability among systems. The strategy is to pay close attention to this area in

acquiring and integrating trusted products and to monitor the status of efforts to produce guidelines and standards.

10. DOD Funding Cutbacks

DOD faces major funding cuts in the Fiscal Year (FY) 1992-1997 period that will eliminate all but the most essential system enhancements. MLS efforts have the potential to reduce costs and also improve mission effectiveness. Because of the development risks and uncertainties currently associated with MLS, however, MLS efforts might also result in increased development costs. In some cases, there might be a high cost to transition to MLS that is more than offset by lower operational costs once the transition is complete.

The strategy is to emphasize cost reduction and return on investment. Particular emphasis will be placed on ensuring that near-term fielding efforts provide acceptable return on investment, so that a favorable climate is maintained in which to pursue longer-term investment in MLS capabilities. Where MLS cannot cut costs, it must be clear -- as with the "smart" technology used in Operation Desert Storm -- that the operational benefits (e.g., improved data fusion) justify the costs.

References

[1] DISA, September 1991, *Target Architecture and Implementation Strategy for the Joint MLS Technology Insertion Program*, Arlington, VA.

[2] DOD, 21 March 1988, "Security Requirements for Automated Information Systems," DOD Directive 5200.28, Washington, DC.

[3] NCSC, 21 October 1988, *GLOSSARY of Computer Security Terms*, NCSC-TG-004, Version-1, Ft. Meade, MD.

[4] NSA, 22 August 1990, "Verdix Corporation VSLAN 5.0 EPL Entry," DOCKMASTER on-line EPL, Ft. Meade, MD.

[5] National Institute of Standards and Technology (NIST), 27 September 1983, *Guideline for Computer Security Certification and Accreditation*, FIPS PUB 102, Gaithersburg, MD.

[6] Curry, D. A., April 1990, *Improving the Security of Your UNIX System*, Information and Telecommunications Sciences and Technology Division (ITSTD)-721-FR-90-21, SRI International, Menlo Park, CA.

[7] AFCEA, 30 April 1989, *The Armed Forces Communications and Electronics Association Information Security Study*, AFCEA.

Implementation Considerations for the Typed Access Matrix Model in a Distributed Environment

Ravi S. Sandhu and Gurpreet S. Suri[1]

Center for Secure Information Systems
&
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

ABSTRACT The typed access matrix (TAM) model was recently defined by Sandhu. TAM combines the strong safety properties for propagation of access rights obtained in Sandhu's Schematic Protection Model, with the natural expressive power of Harrison, Ruzzo, and Ullman's model. In this paper we consider the implementation of TAM in a distributed environment. To this end we propose a simplified version of TAM called Single-Object TAM (SO-TAM). We illustrate the practical expressive power of SO-TAM by showing how the ORCON policy for originator control of documents can be specified in SO-TAM. We provide arguments to support our conjecture that SO-TAM is theoretically as expressive as TAM. We show that SO-TAM has a simple implementation in a typical client-server architecture. Our design is based on access control lists as the principal means for enforcing access to subjects and objects. In addition, certificate servers are introduced for generating certificates for checking access rights in those cases where access control lists are insufficient. A major advantage of our design is that atomicity of operations does not require a distributed commit.

Keywords: Access Matrix, Distributed Systems, Secure Architectures, Access Control Lists, Certificates

1 INTRODUCTION

Distributed systems have become the prevalent mode of computing. Modern systems offer a great deal of flexibility in tailoring a user's environment. The physical distribution of data and other resources can be made as transparent as a user wishes. It is important that security researchers and practitioners provide similar flexibility with respect to access control mechanisms.

To provide flexibility in access control we first need a flexible model which can express a rich variety of security policies. In our opinion flexibility is achieved by allowing users to propagate access rights to other users, with a combination of discretionary and mandatory controls. We would like to give individual users as much discretionary choice as possible, within the constraints required to meet the overall objectives and policies of an organization. For example, members of a project team might be allowed to freely share project documents with each other, but only the project leader is authorized to allow non-members to read project documents.

Security models based on propagation of access rights must confront the safety problem. In its most basic form, the safety question for access control asks: is there a reachable state in which a particular subject possesses a particular right for a specific object? There is an essential conflict between the expressive power of an access control model and tractability of safety analysis. The

[1] The work of both authors is partially supported by National Science Foundation grant CCR-9202270 and National Security Agency contract MDA904-92-C-5141.

access matrix model as formalized by Harrison, Ruzzo, and Ullman (HRU) [5] has very broad expressive power. Unfortunately, HRU also has extremely weak safety properties.

Recently Sandhu [9] has shown how to overcome the negative safety results of HRU by introducing strong typing into the access matrix model. The resulting model is called the Typed Access Matrix (TAM). TAM combines the positive safety results for the Schematic Protection Model [6] with the natural expressive power of HRU.

The safety problem is closely related to the so-called fundamental flaw of discretionary access control (DAC). DAC is vulnerable to Trojan Horses, in part because Trojan Horse laden programs can surreptitiously modify the protection state without explicit instruction from the users. However, even Trojan Horses are constrained by the authorization for propagating access rights. The Trojan Horse vulnerability of DAC does require that we assume the worst case regarding propagation of access rights in a system. What we need therefore is a model, such as TAM, with strong safety properties and broad expressive power.

In addition to balancing expressive power versus safety analysis, a useful model must also be implementable. Our focus in this paper is on implementation considerations for TAM. It is possible to implement TAM as defined in its full generality. However, such a full-blown implementation would be cumbersome and awkward at best. In this paper we identify a simplified version of TAM called Single-Object TAM (SO-TAM). SO-TAM is particularly suited for implementation in a distributed environment. Moreover it retains most, if not all, of the expressive power of TAM. We provide theoretical arguments to support this claim. We also demonstrate how SO-TAM can enforce the ORCON policy for originator control of documents.

The rest of this paper is organized as follows. Section 2 provides a brief review of the TAM model, following which SO-TAM is defined in Section 3. Section 4 expresses the ORCON policy in SO-TAM. This is achieved by taking the ORCON solution of TAM [9], and manipulating it to fit the requirements of SO-TAM. The basic architecture for implementing SO-TAM is discussed in Section 5. Implementation and protocol details of SO-TAM are covered in Section 6. In Section 7 it is then shown how the ORCON example relates to the implementation. Section 8 gives our conclusions.

2 THE TYPED ACCESS MATRIX MODEL

In this section we briefly review the typed access matrix (TAM) model. In a nutshell, TAM is obtained by incorporating strong typing into the model of Harrison, Ruzzo and Ullman [5]. The principal innovation of TAM is to introduce strong typing of subjects and objects. This innovation is adapted from Sandhu's Schematic Protection Model [6].

As one would expect from its name, TAM represents the distribution of rights in the system by an access matrix. The matrix has a row and a column for each subject and a column for each object. Subjects are also considered to be objects. The $[X, Y]$ cell contains rights which subject X possesses for object Y.

Each subject or object is created to be of a specific type, which thereafter cannot be changed. It is important to understand that the types and rights are specified as part of the system definition, and are not predefined in the model. The security administrator specifies the following sets for this purpose:

- a finite set of access *rights* denoted by R, and
- a finite set of *object types* (or simply *types*) denoted by T.

Once these sets are specified they remain fixed (until the security administrator[2] changes their

[2]It should be kept in mind that TAM treats the security administrator as an external entity, rather than as another

definition). For example, $T = \{user, so, file\}$ specifies there are three types, viz., user, security-officer and file. A typical example of rights would be $R = \{r, w, e, o\}$ respectively denoting read, write, execute and own.

The *protection state* (or simply *state*) of a TAM system is given by the four-tuple (OBJ, SUB, t, AM) interpreted as follows:

- OBJ is the set of *objects*.
- SUB is the set of *subjects*, $SUB \subseteq OBJ$.
- $t : OBJ \rightarrow T$, is the *type function* which gives the type of every object.
- AM is the *access matrix*, with a row for every subject and a column for every object. The contents of the $[S, O]$ cell of AM are denoted by $AM[S, O]$. We have $AM[S, O] \subseteq R$.

The rights in the access matrix cells serve two purposes. First, presence of a right, such as $r \in AM[X, Y]$ may authorize X* to perform, say, the read operation on Y. Second, presence of a right, say $o \in AM[X, Y]$ may authorize X to perform some operation which changes the access matrix, e.g., by entering r in $AM[Z, Y]$. In other words, X as the owner of Y can change the matrix so that Z can read Y.

The protection state of the system is changed by means of TAM commands. The security administrator defines a finite set of TAM commands when the system is specified. Each TAM *command* has the following format:

command $\alpha(X_1 : t_1, X_2 : t_2, \ldots, X_k : t_k)$
  if $r_1 \in [X_{s_1}, X_{o_1}] \wedge r_2 \in [X_{s_2}, X_{o_2}] \wedge \ldots \wedge r_m \in [X_{s_m}, X_{o_m}]$
  then $op_1; op_2; \ldots; op_n$
end

or

command $\alpha(X_1 : t_1, X_2 : t_2, \ldots, X_k : t_k)$
  $op_1; op_2; \ldots; op_n$
end

Here α is the *name* of the command; $X_1, X_2, \ldots, X_k$ are *formal parameters* whose types are respectively $t_1, t_2, \ldots, t_k$; $r_1, r_2, \ldots, r_m$ are rights; and $s_1, s_2, \ldots, s_m$ and $o_1, o_2, \ldots, o_m$ are integers between 1 and k. Each op_i is one of the *primitive operations* discussed below. The predicate following the **if** part of the command is called the *condition* of α, and the sequence of operations $op_1; op_2; \ldots; op_n$ is called the *body* of α. If the condition is omitted the command is said to be an *unconditional command*, otherwise it is said to be a *conditional command*.

A TAM command is invoked by substituting actual parameters of the appropriate types for the formal parameters. The condition part of the command is evaluated with respect to its actual parameters. The body is executed only if the condition evaluates to true.

There are six primitive operations in TAM as follows.

enter r **into** $[X_s, X_o]$	**delete** r **from** $[X_s, X_o]$
create subject X_s **of type** t_s	**destroy subject** X_s
create object X_o **of type** t_o	**destroy object** X_o

(a) Monotonic Primitive Operations (b) Non-Monotonic Primitive Operations

subject in the system.

We require that s and o are integers between 1 and k, where k is the number of parameters in the TAM command in whose body the primitive operation occurs.

The **enter** operation enters a right $r \in R$ into an existing cell of the access matrix. The contents of the cell are treated as a set for this purpose, i.e., if the right is already present the cell is not changed. The **enter** operation is said to be *monotonic* because it only adds and does not remove from the access matrix. The **delete** operation has the opposite effect of **enter**. It (possibly) removes a right from a cell of the access matrix. Since each cell is treated as a set, **delete** has no effect if the deleted right does not already exist in the cell. Because **delete** (potentially) removes a right from the access matrix it is said to a *non-monotonic* operation.

The **create subject** and **destroy subject** operations make up a similar monotonic versus non-monotonic pair. The **create subject** operation requires that the subject being created does not previously exist. The **destroy subject** operation similarly requires that the subject being destroyed should exist. Note that if the pre-condition for any **create** or **destroy** operation in the body is false, the entire TAM command has no effect. The **create subject** operation introduces an empty row and column for the newly created subject into the access matrix. The **destroy subject** operation removes the row and column for the destroyed subject from the access matrix. The **create object** and **destroy object** operations are much like their subject counterparts, except that they work on a column-only basis.

To summarize, a system in specified in TAM by defining the following.

1. A set of rights R.
2. A set of types T.
3. A set of state-changing commands.
4. The initial state.

We say that the rights, types and commands define the system *scheme*. Note that once the system scheme is specified by the security administrator it remains fixed thereafter for the life of the system. The system state, however, changes with time.

3 SINGLE-OBJECT TAM

In this section we present a simplified version of TAM called Single-Object TAM (SO-TAM). Our principal motivation in defining SO-TAM is to arrive at a model well-suited to a distributed implementation. We, of course, do not wish to lose or compromise the expressive power of TAM in doing so. We conjecture that SO-TAM is theoretically equivalent to TAM. Arguments in support of this conjecture are given at the end of this section. The natural expressive power of SO-TAM is demonstrated in the next section, where we show how the ORCON policy for originator control of documents is specified in SO-TAM.

The principal restriction in SO-TAM is that all primitive operations in the body of a command are required to operate on a single object. An object is represented as a column in the access matrix. Similarly, when a subject is the "object" of an operation, that subject is viewed as a column in the access matrix. SO-TAM stipulates that all operations in the body of a command are confined to a single column.

Now consider the usual implementation of the access matrix by means of access control lists (ACL's). Each object has an ACL associated with it, representing the information in the column corresponding to that object in the access matrix. The restriction of SO-TAM implies that a single command can modify the ACL of exactly one object. These modifications can therefore be done at the single site where the object resides. This greatly simplifies the protocols for implementing the

commands. In particular, we do not need to be concerned about coordinating the completion of a single command at multiple sites. There is therefore no need for a distributed two-phase commit for SO-TAM commands.

Commands in SO-TAM are further categorized into the following two classes, depending upon the single or multi-object nature of the condition part of the command.

- **Class I**: In these commands the condition part is also single object, i.e., the tests are confined to the ACL of a single object. Unconditional SO-TAM commands also fall into this class. An example of a **Class I** command is given below.

  command $\alpha(S_1 : t_1, O : t_2, S_3 : t_3)$
  if x $\in [S_1, O]$ **then**
  enter z **into** $[S_2, O]$
  end

- **Class II**: In these commands the condition part is multi-object, i.e., the tests require reference to the ACL of more than one object. An example of a **Class II** command is given below.

  command $\alpha(S_1 : t_1, O : t_2, S_3 : t_3)$
  if x $\in [S_1, O] \wedge$ y $\in [S_1, S_3]$ **then**
  enter z **into** $[S_1, O]$
  end

In **Class I** commands the condition and body of the command reference the ACL of a single object. These commands can therefore be executed completely at the site where this single object resides. In **Class II** commands evaluation of the condition part requires reference to the ACL's of several objects. In general these objects can be located at different sites. Various pieces of the condition will need to evaluated at different sites and then combined together. **Class II** commands therefore require a more complex protocol than **Class I** commands. Implementation of **Class I** and **Class II** commands is discussed in section 6.

Now let us consider the expressive power of SO-TAM. SO-TAM with **Class I** commands alone is quite expressive by itself. In particular it subsumes the various transform models of [7, 10, 11]. SO-TAM with **Class II** has very strong expressive powers. As is shown in the next section it can express the ORCON policy. Moreover SO-TAM can easily model the Extended Schematic Protection Model (ESPM) [1, 2]. SO-TAM therefore inherits the theoretical expressive power of ESPM, which is equivalence to the Harrison, Russo and Ullman (HRU) model [5] for the monotonic case (i.e., no **delete** or **destroy** primitive operations). We conjecture that this equivalence of SO-TAM and HRU will also extend to the non-monotonic case. Formal consideration of this matter is beyond the scope of this paper. SO-TAM also inherits the practical expressive power of ESPM demonstrated in [1, 8]. It should be noted that the expressive power of SO-TAM is obtained without compromise on safety analysis.

4 ORCON IN SO-TAM

In this section we demonstrate the expressive power of SO-TAM by specifying an ORCON (originator control) policy [4]. In doing so we also show how multi-object TAM commands can be reduced to single object operations. Specifically we first review the ORCON solution given in [9]. This solution uses multi-object TAM commands. We then show how to construct equivalent SO-TAM commands.

ORCON requires that the creator (i.e., originator) of a document retains control over granting access to the information in the document. For example, let Tom be the creator of an ORCON

	$S_1 : s$	$S_2 : s$	$O : co$	...
$S_1 : s$			own, read, write	
$S_2 : s$				
...				

(a) Subject S_1 creates an ORCON object O

	$S_1 : s$	$S_2 : s$	$O : co$	...
$S_1 : s$			own, read, write	
$S_2 : s$			cread	
...				

(b) S_1 gives S_2 the cread (confined-read) right for O

	$S_1 : s$	$S_2 : s$	$O : co$	$S_3 : cs$	...
$S_1 : s$			own, read, write		
$S_2 : s$			cread		
$S_3 : cs$			read		
...					

(c) S_2, jointly with O, creates the confined subject S_3 to read O

Figure 1: Illustration of the ORCON Policy with multi-object TAM operations

document[3] called SDI. Suppose Tom authorizes Dick to read SDI. The ORCON policy requires that Dick cannot propagate the information in SDI to, say, Harry; either directly by granting Harry read access to SDI, or indirectly by granting Harry read access to a copy of SDI. The prohibition that Dick cannot directly grant read access to Harry is straightforward to enforce. The real challenge for the ORCON policy is how to prevent Dick from copying the information from SDI into some other document, say, SDI-Copy and authorizing Harry to read SDI-Copy.[4]

The ORCON solution given in [9] is based on the ability in TAM to have multiple parents jointly create a child subject.[5] Figure 1(a) shows a fragment of the access matrix in which subject S_1 is the creator (and therefore owner) of object O as indicated by own $\in [S_1, O]$. The notation $S_1 : s$ denotes that S_1 is of type s, and similarly for the names on the other rows and columns. The type of O is co for confined object. In Figure 1(b), S_1 gives S_2 the cread (i.e., confined-read) right for O. This right allows S_2 to create jointly with O subject S_3 of type cs (for confined-subject). This creation results in S_3 getting the child right for S_2 and O. By virtue of being the child of S_2 and O and S_2 possessing the cread right, S_3 obtains the read right for O. This results in the situation shown in Figure 1(c). The scheme will ensure that S_3, by virtue of its type being cs, will never be able to write to any object or create any objects.

The definition of the TAM scheme for this ORCON solution is given below.

[3] An ORCON document is one to which the ORCON policy applies as opposed to, say, ordinary documents to which ORCON does not apply.

[4] Note that Dick as a human being is trusted not to divulge information from SDI to Harry without concurrence of Tom. The problem here is to ensure that Trojan Horse laden subjects executing on behalf of Dick do not surreptitiously leak the information in SDI to Harry.

[5] The solution prohibits subjects spawned by Dick from making copies (or extracts) of SDI. The solution can be extended to allow this with the stipulation that the copies (or extracts) will themselves be originator controlled by Tom.

1. Rights $R = \{\text{own}, \text{read}, \text{write}, \text{cread}\}$
2. Types $T = \{s, cs, co\}$
3. The following TAM commands

(a) **command** create−orcon−object($S_1 : s, O : co$)
create object O **of type** co;
enter {own, read, write} **in** $[S_1, O]$[6]
end

(b) **command** grant−confined−read($S_1 : s, S_2 : s, O : co$)
if own $\in [S_1, O]$ **then enter** cread **in** $[S_2, O]$
end

(c) **command** use−confined−read($S_2 : s, O : co, S_3 : cs$)
if cread $\in [S_2, O]$ **then create subject** S_3 **of type** cs;
enter read **in** $[S_3, O]$
end

(d) **command** destroy−orcon−object($S_1 : s, O : co$)
if own $\in [S_1, O]$ **then destroy object** O
end

(e) **command** revoke−confined−read($S_1 : s, O : co, S_2 : s$)
if own $\in [S_1, O]$ **then delete** cread **from** $[S_2, O]$
end

(f) **command** revoke−read($S_1 : s, O : co, S_3 : cs$)
if own $\in [S_1, O] \wedge$ read $\in [S_3, O]$ **then destroy subject** S_3
end

(g) **command** finish−orcon−read($S_2 : s, O : co, S_3 : cs$)
if cread $\in [S_2, O] \wedge$ read $\in [S_3, O]$ **then destroy subject** S_3
end

Use of the first three commands is illustrated in Figure 1. The remaining commands are for revocation of rights and destruction of objects and subjects.

This scheme is not an SO-TAM scheme, because of command (c) which has multi-object operations. In command (c) subject S_3 has to be created and the ACL of object O has to be modified. In general, this requires the command to execute at two sites contrary to the constraints of SO-TAM. All commands other than (c) are actually **Class I** commands, i.e., single-object condition and operations.[7]

This scheme can be easily converted to SO-TAM. We do this by introducing a parent right. Command (c) is replaced by the following two commands.

(c.1) **command** create−confined−subject($S_2 : s, O : co, S_3 : cs$)
create subject S_3 **of type** cs;
enter parent **in** $[S_2, S_3]$;
enter parent **in** $[O, S_3]$;
end

[6] Strictly speaking this should be written as three separate **enter** operations, one for each of the three rights being entered.

[7] One might question how **destroy subject** is a single site operation, since it requires removal of a row from the access matrix potentially affecting a large number of ACL's. However, we don't need to purge these ACL's immediately in an atomic manner.

(c.2) **command** get−read($S_2 : s, O : co, S_3 : cs$)
if read $\in [S_2, O] \wedge$ parent $\in [S_2, S_3] \wedge$ parent $\in [O, S_3]$
then enter read **in** $[S_3, O]$
end

Figure 2 shows how the scenario of Figure 1 plays out with this modification. In the modified scheme we enter the parent privilege during joint creation by command (c.1). Prior to grant of the read privilege to S_3 the condition in command (c.2) tests for the presence of the parent right. This simple manipulation makes the entire scheme an SO-TAM scheme with single-object operations. Note that command (c.1) is a **Class I** command while command (c.2) is a **Class II** command.

5 THE ARCHITECTURE

In this section we describe a client-server based architecture for implementing SO-TAM. This architecture has evolved from our earlier work [2, 10, 11].

5.1 Global Identifiers

Every subject and object is assigned a type when it gets created. The typing is strong and cannot be altered thereafter. Moreover each subject or object in the system has a globally unique identifier i.e., no two subjects or objects in a system can have the same identifiers. We assume the type of a subject or object is embedded in its identifier. These identifiers have the following structure.

type	identifier

The type field denotes the type of the subject or the object. The identifier field uniquely identifies each subject or object among instances of the same type. Uniqueness of object identifiers reduces to requiring each object to have a unique identifier among instances of the same type. If a particular type is managed by more than one server, uniqueness of the identifier can be ensured by having the following structure.

type	server identifier	identifier

Having made this point, we will use the former global identifier structure in rest of this paper.

5.2 Access Control Lists

Each object in the system is managed by an object server. When the object is a subject, we sometimes call the server a subject server. Each server manages a particular type of object, but the same type of object may be managed by several servers. For example, there may be several file servers in the system. Each object resides at exactly one server.

Each object has an Access Control List (ACL) associated with it. The ACL has the following structure.

	sid1	rights
	sid2	rights
oid	.	.
	.	.
	sidn	rights

	$S_1 : s$	$S_2 : s$	$O : co$	...
$S_1 : s$			own, read, write	
$S_2 : s$				
$O : co$				
...				

(a) Subject S_1 creates an ORCON object O

	$S_1 : s$	$S_2 : s$	$O : co$	...
$S_1 : s$			own, read, write	
$S_2 : s$			cread	
$O : co$				
...				

(b) S_1 gives S_2 the cread (confined-read) right for O

	$S_1 : s$	$S_2 : s$	$O : co$	$S_3 : cs$	...
$S_1 : s$			own, read, write		
$S_2 : s$			cread	parent	
$O : co$				parent	
$S_3 : cs$					
...					

(c) S_2, jointly with O, creates the confined subject S_3

	$S_1 : s$	$S_2 : s$	$O : co$	$S_3 : cs$	...
$S_1 : s$			own, read, write		
$S_2 : s$			cread	parent	
$O : co$				parent	
$S_3 : cs$			read		
...					

(d) S_3 acquires read right for O

Figure 2: Illustration of the ORCON Policy in SO-TAM

To make the construction of the architecture clear we refer to a subject identifier by *sid* and a object identifier by *oid*.

Any access to an object is determined by the rights specified in the ACL for that subject. Similarly all accesses to subjects are dictated by the rights in the ACL possessed by the requesting subject. The ACL's are dynamic in nature and can be manipulated by SO-TAM commands.

5.3 Certificates

In addition each server is associated with a certificate server. The certificate server acts as a mediator for any form of communication between two servers. The certificate server is responsible for creating, encrypting and decrypting certificates for the servers to which it is associated. The certificate generated by a certificate server has the following structure.

oid	Rights	sid

The oid contains the unique identifier for the object in question. The sid is the unique identifier of the subject. The rights field specifies the rights that the subject identified in the sid field has for the object in the oid field.

Since these certificates travel over insecure lines they are made secure by using a public key based encryption algorithm. For this we specify a pair of keys for each server. Out of this pair one of the keys is secret known only to that server's certificate server, while the other one is public and known to all certificate servers. Certificates are doubly encrypted in the usual manner in public-key systems, to ensure their authenticity and confidentiality. They are also time-stamped to avoid replay attacks. Further details are given in the next section. Authentication between users and their servers is assumed. Any authentication protocol from the literature [3] can be employed for this purpose.

6 IMPLEMENTATION OF SO-TAM

The implementation of SO-TAM commands is based on the architecture described in the previous section. All accesses to an object are mediated by the object server responsible for managing that object. Similarly for subject accesses the subject server responsible for that subject mediates the access.

Authentication is also carried out at the time of object/subject access, and must be incorporated into the RPC (Remote Procedure Call) mechanism of the client-server architecture. The servers must authenticate the source of every RPC request. This can be achieved by any of the encryption protocols found in literature [3]. One method would be to provide means for every subject to place its digital signature on every RPC communication to a server. Digital signatures for the reverse communication from object/subject servers to clients can also be incorporated.

We now describe the execution of a primitive operation at a server, followed by protocols for **Class I** and **Class II** commands.

6.1 Primitive Operations

Let us consider each of the primitive operations in turn.

1. **enter x into** $[S_1, O]$

   In this operation the server managing object O enters the x right for subject S_1 into the ACL for object O.

2. **delete x from** $[S_1, O]$

   For this operation the server managing object O deletes the x right that S_1 has from O's ACL. This operation is exactly the opposite of the **enter** operation.

3. **create subject** S_1 **of type** t_1

   The server who will manage subject S_1 creates S_1 with an empty ACL.

4. **destroy subject** S_1

   The server managing S_1 destroys the subject S_1 and discards S_1's ACL.

5. **create object** O **of type** t_2

   The server who will manage object O creates object O with an empty ACL.

6. **destroy object** O

   The server managing O destroys the object O and discards O's ACL.

6.2 Class I Commands

For an unconditional command the server in question simply executes the primitive operations in the body as indicated above. The operations of conditional **Class I** commands are executed only if the specified condition is satisfied. In **Class I** commands the **if** condition can be tested by the server who manages the object in question, simply by reference to the object's ACL.

A typical command with single-object condition verification is shown below.

command $\alpha(S_1 : t_1, O : t_2)$
if x $\in [S_1, O]$
then $op_1; op_2; \ldots; op_n$
end

This command is sent to the server where the listed operations $op_1; op_2; \ldots; op_n$ are to be executed. The command is executed as follows.

1. (a) The server on receiving the request verifies the types of the subjects and objects against the security policy to check the validity of the command. Once the validity is confirmed the server tests the **if** conditional statement. If the command fails the validity tests the request is aborted.
   (b) The server checks the ACL for object O to see if S_1 really possesses the x privilege for O and if so it executes the next step, otherwise the request is aborted.
   (c) If the **if** condition is true the server performs the operations $op_1; op_2; \ldots; op_n$.

6.3 Class II Commands

Verification of the condition in **Class II** commands requires reference to multiple sites. Our protocol for multi-object condition verification is based on inter-server communications. Various pieces of the condition as verified at individual servers and communicated to server A as certificates. Each server has an associated certificate server to generate the certificate.

Consider the following typical example of multi-object verification of a conditional command.

command $\alpha(S_1 : t_1, O : t_2, S_3 : t_3)$
if x $\in [S_1, O] \wedge$ y $\in [S_1, S_3]$
then $op_1; op_2; \ldots; op_n$
end

As specified by the constraints of SO-TAM all the operations $op_1; op_2; \ldots; op_n$ involve only one server. Let us say this is the server for object O and is called server A. In the above command verification of the condition part involves only one additional site, viz., the site of S_3's server. Let us call S_3's server as server B. The protocol is easily extended to additional sites.

In this command, to verify the conditional if statement, server A needs information from the subject server managing subject S_3 as to whether or not S_1 possesses the y right for S_3. This is achieved as follows.

1. (a) Server A checks the security policy to determine the validity of the request. If the validity tests fail the request is aborted.
   (b) Server A checks O's ACL to see whether S_1 possesses the x right for O. If S_1 does indeed possess x for O the command proceeds, otherwise it is aborted.
   (c) Server A further needs information from the subject server managing subject S_3 as to whether or not S_1 has the y right for S_3, so A waits for a certificate from S_3's server. (To prevent A from waiting indefinitely for the certificate to arrive, it waits for a specified period of time and then aborts the command.)
2. (a) Server B, i.e., S_3's server, checks into S_3's ACL to ascertain whether S_1 possesses the y right for S_3. If this is so, B informs B's certificate server to create a certificate and send it to server A. Otherwise server A is notified of a failed condition.
   (b) B's certificate server encrypts the certificate with its own secret key. Then the certificate is again encrypted with the public key of the A's certificate server. The certificate is shown below.

$$(\boxed{S_3 : t_3} \boxed{\text{y}} \boxed{S_1 : t_1} \boxed{TS} \; K_d^B) K_e^A$$

   where K_d^B is the secret key of B (known only to B's certificate server), K_e^A is the public encryption key of A (known to all certificate servers) and TS is a timestamp.
3. (a) When A's certificate server receives the certificate it decodes it in two steps. First it applies A's secret key K_d^A, and it applies B's public key K_e^B. If decryption fails or the timestamp is out of date the request is aborted.
   (b) If the certificate is decoded correctly the information it holds is in the clear and server A has the necessary verification it needs to process the command request.
   (c) If the condition is met server A executes the requested operations op_1; op_2; ...; op_n.

7 IMPLEMENTATION OF ORCON

In this section we give a concrete example of the abstract implementation of section 6 by showing how the ORCON policy of section 4 is enforced.

1. Let Tom be a subject of type s who initiates the following command to create the ORCON object SDI of type *co*.

$$\textbf{command}\ \text{create-orcon-object}(Tom : s, SDI : co)$$

   The kernel of Tom's host, makes a remote procedure call (RPC) to the object server which is responsible for managing ORCON objects created by Tom. This RPC contains the action requested, the sid and oid; all signed under Tom's digital signature. In this instance, the sid = s.Tom and the oid = co.O.

2. On receiving the request the object server authenticates the request originating from Tom. The server then checks the **command** create–orcon–object with respect to its actual parameters to determine its validity. Once the command is determined to be valid, the object server proceeds to create a new ORCON object SDI with the ACL shown below.

co.SDI	s.Tom	own,read,write

The ACL shows Tom to be the owner of the document SDI and possessing own, read and write privileges for it.

3. In this step Tom grants cread (confined read) privilege to Dick (sid = s.Dick). The command is sent to SDI's object server. The request is shown below.

command grant–confined–read($Tom : s, Dick : s, SDI : co$)

The server on receiving the RPC authenticates its origin as Tom. Then it performs the validity checks on the request by checking the sids and oids of the subjects and objects involved in the operation. The server then evaluates the condition part of the command. The server looks into SDI's ACL to see if Tom is the owner of SDI. With this fact confirmed, the **if** condition evaluates to true and the server enters cread privilege for Dick into the ACL, as shown below.

co.SDI	s.Tom	own,read,write
	s.Dick	cread

With this Dick possesses the cread privilege for the confined-object SDI.

4. Now Dick and the object SDI jointly create a new subject Dick′ which is of the type confined-subject (*cs*). The command shown below is sent to the appropriate subject server.

command create–confined–subject($Dick : s, SDI : co, Dick' : cs$)

The subject server on receiving the request authenticates the sender and tests the sids and oids of the subjects and objects involved to determine the validity of the request. Since this is an unconditional command, the subject server proceeds to create a new subject Dick′ with the ACL shown below.

cs.Dick′	s.Dick	parent
	co.SDI	parent

5. Next the read right is obtained by Dick′ via the following command. This command is sent to the object server managing SDI.

command get–read($Dick : s, SDI : co, Dick' : cs$)

Like before the object server makes the authentication and validity tests. Then it checks into its ACL to determine whether Dick possesses the cread privilege for SDI. This information completes one part of the **if** statement. For the other part it relies on information from the subject server managing Dick′.

6. The subject server for Dick′ checks into its ACL to determine whether Dick and SDI are parents of Dick′. Since this is the case, the server informs its certificate server which frames two certificates, shown below, to be sent to SDI's object server.

$$\left(\boxed{\text{s.Dick} \mid \text{parent} \mid \text{cs.Dick}' \mid \text{TS}}\ K_d^B\right)K_e^A$$

$$\left(\boxed{\text{co.SDI} \mid \text{parent} \mid \text{cs.Dick}' \mid \text{TS}}\ K_d^B\right)K_e^A$$

where the K_d^B is the secret key of the subject server for Dick′ and K_e^A is the public encryption key of the object server for SDI. Recall that TS is a timestamp.

7. The certificate server for SDI's object server first applies its secret key K_d^A and then the public key K_e^B of the certificate server for the subject server. Now the certificates are in the clear as shown below.

s.Dick	parent	cs.Dick′

co.SDI	parent	cs.Dick′

Now SDI's object server has complete information to evaluate the condition part of the command. Since the condition evaluates to be true, the server updates the ACL by adding the read right for Dick′ for the ORCON object SDI.

co.SDI	s.Tom	own,read,write
	cs.Dick′	read

Now Dick′ can read SDI but cannot copy it or pass it to another subject (due to Dick′ being a confined subject).

8. Now suppose Tom wants to revoke the read access to Dick′. To do this he issues the following command.

command revoke–read($Tom : s, SDI : co, Dick' : cs$)

The object server for SDI authenticates the command and performs the regular validity tests on the command. With validity of the command confirmed the server checks SDI's ACL to see whether Tom is the owner of SDI and whether Dick′ has the read privilege for it. Since this is true, the server deletes the read privilege for Dick′ for SDI. The purged ACL is shown below.

co.SDI	s.Tom	own,read,write
	cs.Dick′	

Since the read privilege is deleted from the ACL all future accesses by Dick′ to read SDI are denied.

This completes the example.

8 CONCLUSION

In this paper we have considered implementation of the Typed Access Matrix (TAM) model, recently defined by Sandhu [9]. TAM has rich expressive power and yet has strong safety properties. We have defined a simplified version of TAM called Single-Object TAM (SO-TAM). We have shown that SO-TAM has a particularly simple and efficient implementation in a distributed environment. This paper demonstrates how the ORCON policy can be expressed in SO-TAM and implemented in

the architecture. We conjecture that SO-TAM has the same expressive power as TAM. Theoretical arguments in support of this conjecture have been provided.

The implementation is based on an architecture which makes use of both access control lists and certificates. All accesses to subjects and objects are mediated by subject and object servers respectively. Access control lists are used for this purpose. Each server in addition has a certificate server under its domain. The certificate server has the function of creating and decrypting certificates used for communications between servers over a potentially hostile network.

References

[1] Ammann, P.E. and Sandhu, R.S. "The Extended Schematic Protection Model." *Journal of Computer Security*, to appear.

[2] Ammann, P.E., Sandhu, R.S. and Suri, G.S. "A Distributed Implementation of the Extended Schematic Protection Model." *Seventh Annual Computer Security Applications Conference*, 1991, pages 152-164.

[3] Davies, D.W. and Price, W.L. *Security in Computer Networks.* John Wiley & Sons (1989).

[4] Director of Central Intelligence Directive No. 1/7 "Control of Dissemination of Intelligence Information," 4 May 1981.

[5] Harrison, M.H., Ruzzo, W.L. and Ullman, J.D. "Protection in Operating Systems." *Communications of ACM* 19(8), 1976, pages 461-471.

[6] Sandhu, R.S. "The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes." *Journal of ACM* 35(2), 1988, pages 404-432.

[7] Sandhu, R.S. "Transformation of Access Rights." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1989, pages 259-268.

[8] Sandhu, R.S. "Expressive Power of the The Schematic Protection Model." *Journal of Computer Security*, Volume 1, Number 1, 1992, pages 59-98.

[9] Sandhu, R.S. "The Typed Access Matrix Model" *IEEE Symposium on Research in Security and Privacy*, Oakland, CA. 1992, pages 122-136.

[10] Sandhu, R.S. and Suri, G.S. "A Distributed Implementation of the Transform Model" *14th National Computer Security Conference*, Washington, DC, October 1991, pages 177-187.

[11] Sandhu, R.S. and Suri, G.S. "Non-Monotonic Transformation of Access Rights" *IEEE Symposium on Research in Security and Privacy*, Oakland, CA. 1992, pages 148-161.

IMPLICATIONS OF MONOINSTANTIATION IN A NORMALLY POLYINSTANTIATED MULTILEVEL SECURE DATABASE

Frank E. Kramer

Steven M. Heffern

MLS GDSS PROGRAM

Digitial Equipment Corporation

721 Emerson Rd. P.O. Box 227320

St. Louis, MO

Keywords: Polyinstantiation, Multilevel Security, Relational Database

Point of Contact: Frank E. Kramer (314) 991-6268

Abstract

The intentional use of polyinstantiation within the context of a relational database model is a useful mechanism for the incorporation of data entered at differing security levels into a single multi-level relational database. However, there may be some data entities in a given application where the polyinstantiation of that data entity could be in conflict with operational requirements and must be relaxed. In these cases, the monoinstantiation of of that data entity may be mandated in an otherwise polyinstantiated database. Within the MLS Global Decision Support System (MLS/GDSS) system being constructed for the United States Transportation Command/Air Mobility Command (USTRANSCOM/AMC), the treatment of textual remarks calls for such a monoinstantiation due to the operational requirement that remarks at all security levels dominated by the user's security level be made available and immediately viewable to the user. For most of this application, field-level labelling is accomplished by the collapsing of polyinstantiated tuples into a multilevel tuple for presentation to the user interface. This allows the user to view the highest level version of that data for which he is cleared. For textual remarks, each remark, regardless of security level, is considered important and should not be overlayed by information at a higher security level.. Each remark in the database must then be given a unique key value such that a polyinstantiation will not occur when it is stored in the database. The implications of this monoinstantiation and the special procedures required in the determination of system-generated database keys for these data are discussed.

Introduction

The Air Mobility Command (AMC, formerly the Military Airlift Command) uses the Global Decision Support System (GDSS) for planning, allocating, scheduling, and controlling the nation's airlift capabilities to support Department of Defense requirements. The Command and Control Multi-Level Security Program (C2 MLSP) was initiated to demonstrate the operational capabilities produced when applying Multi-Level Secure technology to the existing GDSS system.

Digital Equiment Corporation is the system integrator responsible for re-hosting the GDSS system to a Multilevel Secure system certifiable to security class B1 or better (NCSC Orange Book) using commercial off-the-shelf (COTS) products. The retro-fitted system, MLS/GDSS, is capable of supporting UNCLASSIFIED[1] through SECRET information. Among AMC's requirements for this system are:

1. All information at all security levels (UNCLASSIFIED through SECRET) will be stored in a single database. Controlled access to information contained in this database must be enforced by the TCB and its extensions and is determined by the user's clearance level.
2. MLS/GDSS will support the concept of cover stories for classified information.
3. MLS/GDSS will retain the 'look and feel' of the existing, single-level GDSS in order to reuse existing software to the greatest extent and to minimize the retraining of personnel.
4. Sensitivity labels must be applied at the data element (field) level. Current MLS COTS DBMSs label data at the tuple level and not at the data-element level. Therefore, the TCB was extended to include the means to label individual data items in a composite tuple presented to the application.
5. Sensitivity labels of displayed data must be available for display. Users must be able to view readily the sensitivity labels of all displayed data items in order to determine whether that data may or may not be disclosed.

It was discovered during the design and implementation of MLS/GDSS that the model used for the data architecture [1], when viewed in the light of user requirements, was insufficient to process and maintain certain types of information.

It must be stressed that the design for this system is driven. to a great degree, by item 3, above. The users of the application were, for the most part, flexible in their acceptance of the constraints imposed with the re-hosting of the system for multilevel security. However, changes in functionality that would result in changes to the business rules for operations were carefully

[1] In the context of this report, the term "unclassified" is synonomous with Sensitive Unclassified. The term "uncleared user" is a user who has no clearance and is authorized access to only Sensitive Unclassified data. A "cleared user" is one who is cleared to view information classified up to the Secret level. The term "low" refers to the Sensitive Unclassified sensitivity level. The term "high" refers to the Secret sensitivity level. The letter "U" is the abbreviation for Sensitive Unclassified data. The letter "S" is the abbreviation for Secret data.

weighed against the risks associated with maintaining the functionality of the current, single level, system. In many instances, these risks were determined to be more acceptable than the functional changes that would result from the removal of those risks.

Polyinstantiation

A polyinstantiated database is a database that can contain tuples containing the same primary key information, but holding different versions of non-key data at different sensitivity levels . The use of a polyinstantiated data architecture is recommended whenever a *logical* tuple must contain data with differing sensitivities, i.e. tuples with field-level labeling. With MLS/GDSS, all database tuples contain some fields that are never classified and are necessary for AMC personnel operating at UNCLASSIFIED security levels to function. Because all currently released COTS MLS DBMSs label information at the tuple level, it is necessary to generate a multilevel tuple from a collection of single level tuples for persentation to the user. It was determined that the most efficient means for providing this capability was a database design involving intentional polyinstantiation wherby an existing UNCLASSIFED tuple is polyinstantiated at a higher level whenever one or more data items within that tuple become classified [1].

Polyinstantiation is also utilized in order to enable the generation and storage of cover story information to be presented to the uncleared user. In this context, a cover story is a version of information presented to the uncleared user that hides the existence of classified information, thus preventing unwanted inferences. For example, uncleared flight line personnel need to know the true location and time of an arriving flight in order to service that flight. These data items may not be classified or even classifiable. But these personnel do *not* need to know cargo and passenger details or the intended departure destination. If the latter data items are classified, cover stories serve to conceal these details and prevent the inference of the existence of a classified version of the information. The creation of UNCLASSIFIED tuples with cover stories for these data, and SECRET tuples containing the actual information, with both tuples having the same key field data, defines the polyinstantiation.

In the MLS/GDSS effort, the UNCLASSIFED tuple is stored in the database first, and serves as the base record for information that may be entered at higher sensitivity levels. Users logged in to a SECRET session may then modifiy values in these tuples. These modifications cause the tuples to be polyinstantiated at the SECRET level if existing tuples do not already exist at that level. When the SECRET user reads this information, the system collapses the polyinstantiated tuples and overwrites the UNCLASSIFIED information with the SECRET information, thus presenting a multilevel tuple for display.

Obviously then, the use of polyinstantiation enables the modification of lower sensitivity level data with information at a higher sensitivity level while maintaining the lower level data for access by the uncleared user. Implicit in this is the assumption that information with the higher sensitivity also has higher integrity.

Monoinstantiation of User Remarks

There are, however, data for which the level of integrity is considered to be the same for multiple levels of security. In these cases, the more sensitive information is considered to be either

an extension to or conceptually separate from the lower sensitivity information. As a result, polyinstantiation and the subsequent collapse of data should be considered invalid because the information at the lower sensitivity level should not be overwritten (as by the collapsing mechanism) by data at the higher level.

In the MLS/GDSS effort, this situation occurs with textual remarks entered by users. The AMC has mandated that textual remarks not be collapsed in the MLS/GDSS system in order that a user be able to view, simultaneously, all related remarks stored in the database, not only at his current process security level, but those stored at lower levels as well. This means that a remark field cannot be stored as a field in a polyinstantiated tuple, which may be overwritten by higher sensitivity information when the tuple is read and a multilevel tuple is constructed. It must be stored in a separate REMARKS table which is mono-instantiated (polyinstantiation has been turned off) and, therefore, will not be subject to collapse. The polyinstantiated database record where the remark would normally be stored is referred to as the "parent record" for the REMARKS table tuple.

The monoinstantiation of remarks has an important effect on functionality and user friendliness. Without polyinstantiation, updates to a tuple take the form of replacement rather than annexation. This replacement is different from the overwriting that occurs in the collapse of polyinstantiated tuples; with collapse, only those fields that contain higher sensitivity data are overwritten, whereas with replacement, all fields are effectively overwritten due to the rise in sensitivity label for the entire tuple. In the case of remarks, if a user logged in at a SECRET level modifies an UNCLASSIFIED remark, the subsequently stored remark will replace the existing UNCLASSIFIED remark and the sensitivity label will rise to SECRET. The remark record will then be unavailable to the UNCLASSIFIED user. In some parts of an application, this would be the desired result, while in others the operational functionality must be modified to disallow editing of a remark which has a sensitivity label different from that of the level of the user's process.

There are several other characteristics of remarks data, driven by AMC user requirements, that mandate special handling characteristics. It would perhaps be instructive to describe these by way of an example taken from the current, single level, GDSS system. Figure 1 shows a part of a GDSS screen form which displays both scheduled and actual arrival and departure information for the AMC mission AAM183602192, as well as any remarks which were made pertaining to events occurring during the mission. Briefly, the columns in the schedule data portion of the form have the following meanings:

- MISSION NBR - Mission number designation.
- CT - Crew type
- ICAO - Four character airfield identifier
- C - Purpose code
- STA - Status code, either arrival or departure.
- TIME, ATA/D - Scheduled and actual time for the event in Julian-day/24-hour-time format.
- RM - Remark sequence number. Points to remarks sequence at bottom of form.
- A - Advisory reason code

- D - Delay reason code
- DLY - Delay time
- TAIL# - Aircraft assigned to mission
- ADD - Additional crew flag.

Each line in the schedule data portion corresponds to an event. The events shown in Fig. 1 are scheduled arrival (ARR), scheduled departure (DEP), arrival at diverted (unscheduled) airfield (ADV) and departure from diverted airfield (DDV). For six of these events, remarks that were entered by flight controllers describe or explain some details associated with those events, and for a particular remark, the corresponding database EVENTS record is the parent record for that remark.

The value in the RM column for a particular event in the schedule data portion of the form identifies the sequence number for the remark or remarks entered for that event. This number reflects the chronological order in which an event was first associated with a remark, and is displayed as the integer portion of the number preceeding each remark. It may be seen in Fig. 1 that for those events with remark sequence numbers 2 and 5, there is more than one remark associated. These remarks are "sibling remarks" and are differentiated chronologically by the number following the decimal, the secondary sequence number. This secondary sequence number will be incremented with each new remark generated for a particular event, with the exception of the actual arrival or departure controller's remark, which is always given the secondary sequence number of zero. In order to maintain database integrity, the sequence identifier, consisting of the sequence number and secondary sequence number, along with parent record information are stored as key fields in the REMARKS tuple containing the text of the remark.

Users of the GDSS system have specified three requirements specific to these sequence identifiers:

1. They must be chronologically relevant. The primary sequence numbers for remarks associated with events must reflect the order in which the first remark for an event was entered. Secondary sequence numbers must reflect the order in which remarks were entered within an event, with the exception of arrival/departure controller's remarks which are always zero.
2. They must be constant in time. These numbers are used for internal communications by AMC personnel in referring to specific remarks. They cannot increment due subsequent update of the remark or decrement by the deletion of preceeding remarks.
3. They must remain constant across security levels. These numbers are used by AMC personnel in communicating with each other. The sequence/secondary sequence composite must be the same for a particular remark regardless of the user's session security level.

The above requirements then dictate that remark sequence identifiers must be strictly coupled with the remark text, and so cannot be determined at run-time and must be associated with the remark record in the database at the time of commitment. Because the user operating at an UNCLASSIED security level does not have read-access to those remarks and associated sequence identifiers that are CLASSIFIED, the sequence identifiers cannot be assigned interactively by the user if database integrity is to be maintained, and must, therefore, be generated automati-

cally by the system whenever a new remark is committed to the database. It is in this generation that data integrity and system security are in opposition.

Existence Checking and Trusted Read-Up Capability

Whenever a new remark is committed to the database, a sequence identifier for that remark, used as part of the database key, must be determined by the system on behalf of the user. Because of AMC user requirements, as outlined above, and the monoinstantiated nature of the REMARKS table, the sequence identifier must be unique with respect to a given parent record and must reflect the relative time of entry for the remark. In order for the new sequence identifier to be calculated properly by the system, any process committing a remark to the database must have access to the sequence identifiers for *all* remark records which are related to the committing remark, *regardless of sensitivity level.*

To achieve this functionality, a trusted procedure was developed to perform a very specific read-up function. This procedure temporarily gives the user process read capabilities at system high on the REMARKS table. This procedure is necessary for the obtention of all remark unique sequence numbers for a given parent record, and the calculation of the system-generated key values for the new remark [2]. The function returns an integer which represents the highest sequence number or secondary sequence number found in the returned remarks records. This function cannot query any records other than those in the REMARKS table and it does not return, or make available, any textual remark information to the untrusted application. The application then uses this integer to determine the proper sequence identifier for the new remark prior to storage in the database.

Covert and Inference Channels

It is recognized that the above procedure produces a mechanism whereby a covert channel may be exploited. In addition, because the sequence identifiers for a remark are unique for a parent record, regardless of sensitivity level, it is possible for the UNCLASSIFIED user to infer that CLASSIFIED remarks might exist for that parent record when he observes "gaps" in the sequence identifiers on the screen form. This inference channel is significantly narrowed by the fact that remarks are routinely deleted, so providing gaps *within* a sensitivity level. This is illustrated in Fig. 1 where there is a gap between unclassified remarks 2.1 and 2.3 caused by the deleteion of remark 2.2 . Thus, a gap in sequence does not necessarily imply the existance of higher-level data, but may simply reflect the deletion of data from the database. The introduction of this risk was deemed acceptable by the AMC when weighed against the business rule changes that would be forced were this mechanism not in place.

Summary

The use of moninstantiation in a normally polyinstantiated database can serve a useful function if information classified at lower levels is not intended to be replaced by information classified a higher levels, but is considered to be of equal integrity and should not be hidden from the user at a higher classification.

References

[1] Doncaster, S., Endsley, M., and Factor, G. 1990. "Rehosting Existing Command and Control Systems Into a Multilevel Secure Environment". Proc. Sixth Annual Computer Security Conference. Tucson, Arizona

[2] Nelson, D. and Paradise, C. 1991. "Using Polyinstantiation to Develop an MLS Application". Proc. Seventh Annual Computer Security Conference. San Antonio, Texas.

```
============SCHEDULE============          ==============ACTUAL===============

 MISSION NBR    CT ICAO C STA TIME       RM A D DLY TAIL# ICAO C  ATA/D    ADD
-------------   -- ---- - --- --------   -- - - --- ----- ---- -  -------  ---

AAM183602192       KCHS P DEP 192/1800    1          70014 KCHS P 192/1723  Y
                   KSSC P ARR 192/1830                     KSSC P 192/1755
AAM183602192       KSSC O DEP 193/1200                70014 KSSC O 193/1203
                   YAR1 R ARR 193/1345    3                    *
AAM183602192    B  YAR1 R DEP 193/1445    2                    *
                   EDAF C ARR 193/2115                     EDAF C 193/2204
AAM183602192       EDAF R DEP 194/1730                70014 EDAF R 194/2033  Y
                   KBGR R ARR 195/0155    4                    *
AAM183602192       KBGR R DEP 194/0410                     *
                         ADV              5                KWRI K 195/0615
AAM183602192             DDV                        70014 KWRI K 195/0900
                   KSSC U ARR 195/0655    6                KSSC U 195/0900

================================== REMARKS ==============================
 6.1 MISSION WILL OPERATE ON A PERMIT TO PROCEED FROM KWRI; NEEDS CUSTOMS AND
     AG AT KSSC. ROUGH ETA KSSC WILL BE 195/0800Z. KWRI/PALMER. KSSC/HESS,
     : WRI/RAGAN 195/0630
 5.2 RETRANS FOR IPS: WRI/FULLAN 195/0542 : WRI WELLIS 195/0546
 5.1 DVRT DUE TO LACK OF CREW DUTY DAY TO CONTINUE TO KSSC. ADVISED
     KWRI/FULLEN
     OF DVRT AND REQUEST CUSTOMS, AG, AND IMMIGRATION. : WRI/MCCALL 195/0431
 5.0 106 ENROUTE WINDS REQ FUEL: WRI/WELLIS 195/0558
 4.0 OVERFLY DUE TO THUNDERSTORM ACTIVITY. ACFT WILL DVRT TO KWRI FOR FUEL
     AND
     WILL RON DUE TO LACK OF CREW DUTY DAY TO CONTINUE TO KSSC : WRI/MCCALL
     195/0429
 2.3 AR COMPLETE ON LOAD 41K. MAC/ANDERSON : WRI/HARVEY 193/1716
 2.1 TALKED TO ACFT. ACFT STATES HE WILL BE APPROX 10 MIN LACT TO ARCT,
     KPSM-157ARG, AMN CUMMINGS NOTIFIED : WRI HARVEY 193/1312
 3.0 AR : RMS/WESTON 194/2232
 1.1 EARLY ARRIVAL AT KSSC APPROVED PER "AH" : HRT/KCHS_MORROW 192/1412
```

Figure 1. Example GDSS screen form showing flight schedule data and event remarks.

INFORMATION SYSTEM SECURITY ENGINEERING: CORNERSTONE TO THE FUTURE

Dr. Donald M. Howe
National Security Agency
Fort George G. Meade, Maryland 20755-6000
(410) 684-7295

Abstract

This paper describes current work toward identifying and practicing a system security engineering methodology. Significant results and accomplishments are discussed based on experiences during the formative stages of this work. The work can be characterized as an attempt to provide the cornerstone methodology that will serve as a structured building block to the future. One of the primary objectives is to incorporate evaluation considerations at an early system development stage to reduce risk.

Keywords: *security engineering, systems, evaluation*

1. Introduction

Secure information processing and telecommunications system requirements are increasing throughout society. As intrinsic faults, viruses, worms and other successful malicious attempts become more serious and evident, there could be an exponential increase in the demand for security (with emphasis on integrity) for our information systems. Unfortunately, there is no known way to insure security, especially in distributed systems, unless every detail of performance and potential access is known and effective countermeasures are rigorously instituted and observed.

A dilemma faced today is the amount of evaluation resources necessary to verify the level of security provided by each component of a system. That dilemma is compounded by an imprecise notion of the consequences of combining trusted components into a system, and further complicated by the introduction of untrusted components. This leads to the conclusion that it is too difficult to build a truly secure system, or alternatively, that it will take forever to identify what is not known about the potential faults. When faced with this ominous conclusion, conventional wisdom indicates that one should identify tradeoffs based on what we do know and attempt to quantify expectations and limitations. The methodology for identifying and practicing secure information system composition and evaluation is broadly defined in this paper to be security engineering.

Security engineering is intended to be precise in that it is a structured approach to composing and evaluating systems based on what we have learned through experience, that is, it is empirically based (as is any true engineering approach). Theories become evident as a result of the coalescing of the appropriate experiences, but the theory, fortunately or unfortunately, does not come first. This paper summarizes the findings, to date, of an intense growth, currently embryonic, effort that is directed toward quantitatively structuring secure information system composition and evaluation. The effort is expected to be a continuing evolutionary process, with reporting and feed-back plateaus, representing specific building blocks to the future.

The pivotal principle in system security engineering is the adherence to standards and criteria published by credible organizations after comprehensive review. For example, the absence of a standard for interfaces involving security headers or labels between two components of a system will likely result in a unique software or hardware conversion/patch that could potentially be the source of a security fault. In some cases standards can provide consistent opportunities to exploit security faults. In general, however, it appears that there is less overall vulnerability if standards are followed.

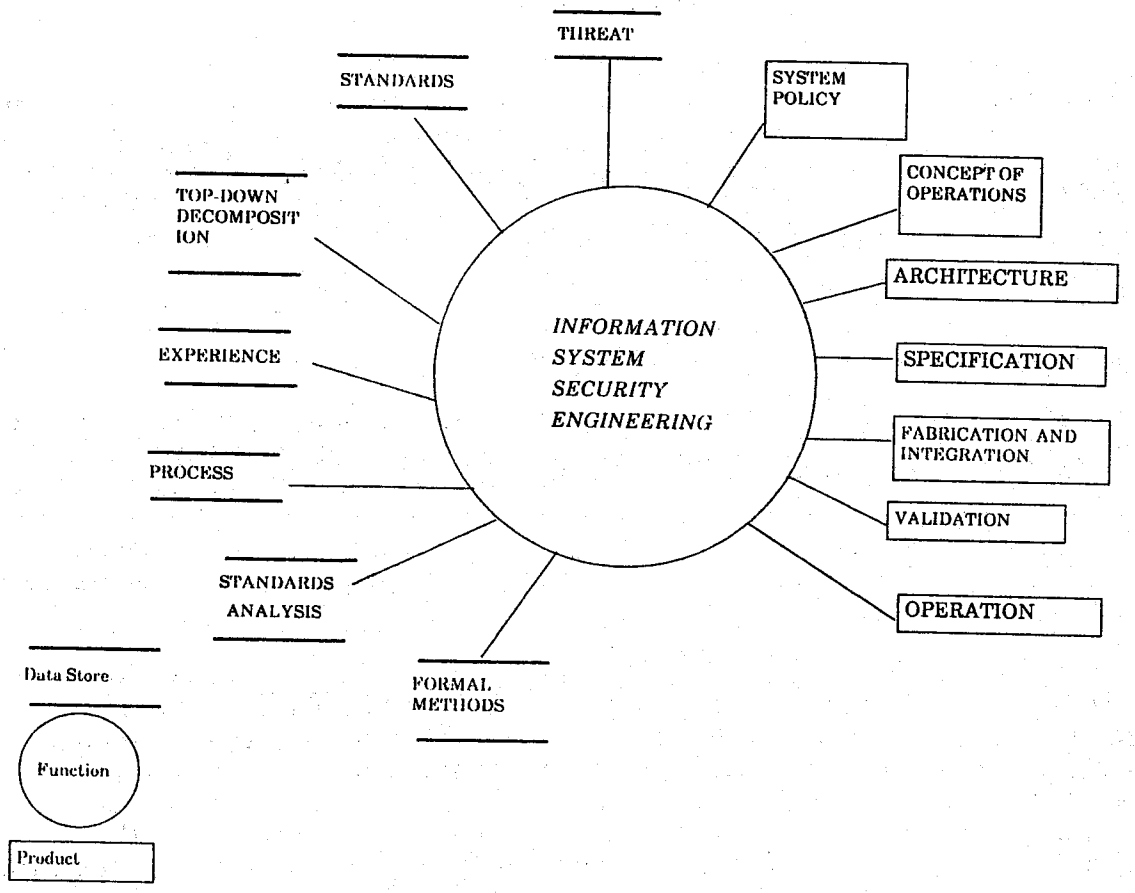


Figure 1. Context Diagram for Security Engineering Principles

Other security engineering principles for the composition and evaluation of systems are based on observations by experts. These include: 1.) security attribute identification and incorporation facilitated through a top-down decomposition analysis, 2.) maintenance of a working knowledge of related system experience within a framework, 3.) adherence to a rigorous process, employing practical guidelines, incorporating extensive evaluation and feedback within all development stages 4.) employment of structured analysis tools to facilitate mission function requirements understanding and realization, together with security as an integral mission requirement and 5.) formal proofs, to the extent practical. Figure 1. illustrates the security engineering principles in a context diagram. This paper provides details of the information system security engineering principles and identifies results based on applications.

2. Background

In January 1992, the Washington Post reported that the Iraq Air Defense System control screens were essentially rendered useless during Desert Storm by a system component modification made during shipment. In mid 1991 telephone cable and switch problems occurred in the Washington D.C., Los Angeles, Pittsburgh, and San Francisco areas. On 15 January 1990, AT&T collapsed due to a flaw in the design of the switch recovery algorithm. On 2-3 November 1988 the INTERNET worm was released resulting in severe degradation. The Desert Storm case highlights the potential security problems associated with uncontrolled distribution of hardware. The three significant 1991 telephone problems were traced to: an untested code patch, faulty signaling protocol implementation, and lack of alarm recognition. The INTERNET worm exploited inherent weaknesses in system software passwords and networking software.

These examples lead to the conclusion that disastrous consequences are possible if a concerted attack is initiated to exploit vulnerabilities of software based systems. Similarly, strategic advantage, tactical battles, and wars are won based on intelligence obtained from information channels that appear secure. Perhaps more importantly, at least for the purpose of this paper, the examples illustrate that vulnerabilities can not be easily contained; every step from system concept formulation to successful system operation can have a critical security fault. Information security is a system problem.

System problems lead to an emphasis on system solutions. The systems approach relies on a working technical knowledge of component security and reliability vulnerabilities, together with knowledge of methods to reduce the potential of a successful threat in a specific application. Security must be viewed as an integrated attribute of the overall system mission. The systems security engineering approach employs the fundamentals of any successful project: good planning, thorough design, sound implementation, reasonable verification, and sensible operation.

3. Security Engineering Principles.

3.1. Standards: Explicit agreement on the definition of criteria, key words, acronyms and concepts is imperative for success. The Department of Defense Trusted Computer System Evaluation Criteria, National Computer Security Center Trusted Network Interpretation, and Air Force Trusted Critical Computer System Evaluation Criteria are examples of publications that provide a common basis for implementation and

evaluation. In some cases, however, there are gaps in specific publications that result in evaluation ambiguities. Ideally, criteria should be published in a way that is very explicit, permitting self evaluation by the designer/ developer (a lofty goal). Security engineering has the responsibility (and challenge) to quantify known publication gaps, that may be subject to interpretation, so that the designer and evaluator recognize the tradeoffs inherent in specific component/system capability. References [2]-[12] indicate the core publications and criteria for security engineering that provides a common ground for designers and evaluators. Reference [1] provides a comparison of the published and draft criteria, together with distributed system applicability, gaps, and recommended extensions.

The development of a unified information system security criteria (INFOSEC criteria that encompasses distributed multilevel security) is a necessary but perhaps currently elusive goal because of the intrinsic complexity and technology available for the foreseeable future. Technology and expectations of technology are advancing very rapidly. Robust application of published (or credible draft/proposed) criteria and standards. in a structured-engineering-tradeoff manner, may be the best approach at the present time.

The National Security Agency (NSA) publishes a quarterly compilation of information system security products and services. The publication [19] lists the NSA-evaluated information systems security products and services that may be used to protect information at several levels of sensitivity. It is an essential working aid for designers of systems having security needs.

The open publication and acceptance of hardware and software security standards, together with the respective evaluation criteria, is critical to the reduction, and hopefully elimination, of potential system security faults. There are security standards published by the International Organization for Standardization (ISO), International Telegraph and Telephone Consultative Committee (CCITT), and the Institute of Electronic and Electrical Engineers(IEEE). The most important standard is the ISO Open Systems Interconnection Basic Reference Model Part 2, ISO 7498-2-1988 (E). It contains detailed descriptions of security services, mechanisms and layers. The ISO 7498-2 appendices provide background, policy information and justification for security service and mechanism placement. Standards are required for interoperable networking and security, and distributed security.

In addition to published definitions, criteria and standards there are specific concepts, recognized by the computer security community, that form a basis for design and evaluation. These concepts are the Bell and LaPadula [13] model for multilevel security, and for integrity: the Biba [12] and Clark and Wilson [14] models. The Reference Monitor Concept [2] and associated Security kernel are important. Access control concepts including Type-enforcement [21] are also important.

3.2. Top-Down System Architecture Decomposition: Within the Department of Defense (DOD), INFOrmation SECurity (INFOSEC) consists of an integrated concept encapsulating communications, computer, transmission and operations security. The goal of information system security engineering is to cost-effectively address all aspects of INFOSEC.

Within the computer security community, the generally accepted attributes of information security are the preservation of confidentiality, integrity and availability. Many believe that these three attributes are incomplete. Parker [15] proposed

the addition of utility (fitness for a purpose) and authenticity (conformance to fact) attributes. Others have proposed the addition of accountability and assurance. The Defense-Wide Information Systems Security Program (DISSP) [16] identified nine system security architecture attributes (actually a mixture of attributes and mechanisms) and two operational categories (interoperability and performance).

The DISSP system security attributes include: 1) Physical, Procedural and Personal Security, 2) Confidentiality, 3) Accountability, 4) Authentication, 5) Access Control, 6) Integrity, 7) Nonrepudiation, 8) Availability, and 9) Assurance. These security attributes, combined with the International Organization for Standardization (ISO) communications layers and basic system elements provide a three-dimensional matrix that is useful for characterizing systems. The DISSP framework (attributes, definitions, and matrix layered components) has been successfully employed for surveying and describing the security of five existing systems/programs and is being used to analyze additional systems. The attributes are useful as a top level information system security check-list to facilitate security policy preparation and review. The framework has been successfully used for mapping existing system policy, captured through the attributes, to implementation as system security mechanisms. Efforts are being initiated to determine the usefulness of the framework as an integral part of new system development.

3.3. Related System Knowledge: There is a wealth of experience available in the development of systems that are identified as having a system high security environment. Experiential knowledge of these systems has limited value for today and tomorrow's problems. There is very little experience in distributed multilevel secure network applications. Siil [20] provides one of the first experience based results of the relatively new and unexplored territory of Multi-Level Secure (MLS) networking. Siil describes the distributed auditing and label management issues, together with lessons learned from experience, for porting AT&T's B1 rated system V/MLS operating system to the AT&T 3B4000 super-minicomputer. The experience indicates the feasibility of a specific MLS network application, without significant performance degradation.

There have been unsuccessful secure distributed system attempts and there are a few systems that are currently in the design stage (THETA, DTMach, and Secure Alpha). Understanding the experiences and lessons learned from abandoned and currently viable candidate systems provides a valuable knowledge base for system security engineering. Rules for composing systems can be obtained from experience.

3.4 Process: A successful system is the result of hard work by motivated, knowledgeable individuals The standard steps in the development process include: mission identification, concept formulation, function specification, threat analysis, policy definition, vulnerability and risk analysis, architecture selection, concept of operations preparation, design/specification, fabrication/production/integration, installation, accreditation, and operation. Guidelines are being prepared to facilitate the incorporation of security into each step of the process. Security retrofits for systems should follow the same process; the primary difference being addition or modification vice comprehensive integration in the early stages of system development.

The ageless axioms of system development apply: 1.) it is cost-effective to recognize problems and take corrective action early in the process; 2.) evaluation is an intrinsic consideration in each step; 3.) communication, feed-back, correction, and iteration are essential; 4.) attributes (e.g. security) are comprehensively integrated into the overall mission and functions; 5.) management is committed.

Risk assessment, together with cost-benefit analyses, are pivotal to making intelligent tradeoffs throughout the process. Weiss [18] describes a method to derive a cost-effective system security architecture and integrate it into the system design process. Efforts are currently underway to apply these and related concepts. Gathering data for this type risk analysis can be somewhat overwhelming and tedious. Unfortunately, there does not appear to be a better or easier way to confidently make cost effective tradeoffs. The specific application and given constraints should be used to determine the extent of the risk assessment effort to be undertaken.

3.5 Structured Analysis: This is an extremely useful technique for identifying the details of system requirements in a structured, quantitative, manner. Products include: context diagrams, data flow diagrams, data dictionaries, state transition tables and other very useful definitive system material. Structured analysis can be computer based or done using paper and pencil. There are several Computer Aided Software/System Engineering (CASE) tools available. Each has an individual advantage. Regardless of the tool selected (even paper and pencil), the benefits of forced requirements definition employing common terms by all interested parties is extremely valuable to the success of the initial stages of the process. Candidate CASE tools are being evaluated at the present time.

3.6 Formal Methods: Applying formal methods to distributed systems appears to be an abstract intellectual exercise at the current time. This does not, however, detract from the need to find practical methods to apply formal evaluations to systems. Research is continuing but it is unlikely to yield significant results in the near future.

4. Applications

There are six pilot projects in various stages of development that are employing parts of the methodology described above. The first project is a major communications support system. Security was incorporated in the early stages of mission definition. Detailed security requirements are included in the overall statement of mission requirements. A security policy has been written. Structured analysis, including context diagrams, are being used. A risk analysis has been completed. The next step is the selection of an architecture. Progress and results appear encouraging. The other projects are in earlier stages of development. Each project is making reasonable progress.

5. Conclusions

This paper has presented a systems security engineering overview. The work is an attempt to provide a development structure that incorporates evaluation considerations at an early stage and continues throughout the development process. The results from the embryonic stage of this effort are very encouraging.

Acknowledgement

This paper provided information system security engineering insights, together with an overview of work currently underway in the Information System Security Engineering Office at the National Security Agency. Many of the office personnel have provided significant inputs. In particular, Bruce Bottomley, George Stephens, Russell Flowers, Paul Boudra, Carl Cecere, Devolyn Arnold, Harold Staton, and Mike Sheridan have made valuable contributions.

References

1. P.G. Neumann, N.E. Proctor, PREVENTING SECURITY MISUSE IN DISTRIBUTED SYSTEMS. Prepared for Rome Laboratory under contract F30602-90-C-0038, MARCH 20 1992.

2. DOD, TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA. Department of Defense Standard 5200.28-STD, December 26,1985.(Orange book)

3. NCSC, TRUSTED NETWORK INTERPRETATION (TNI). National Computer Security Center, NCSC-TG-005 Version-1, 31 July 1987.(Red book)

4. NCSC, TRUSTED NETWORK INTERPRETATION ENVIRONMENTS GUIDELINE. National Computer Security Center, NCSC-TG-011 Version-1, 1 August 1990.

5. NCSC, GLOSSARY OF COMPUTER SECURITY TERMS. National Computer Security Center, NCSC-TG-004 Version-1, 21 October 1988.

6. NCSC, TRUSTED DATABASE MANAGEMENT SYSTEM INTERPRETATION OF THE TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA (TDI). National Computer Security Center, NCSC-TG-21, Version-1, April 1991.

7. NCSC, GUIDANCE FOR APPLYING THE TRUSTED COMPUTER SECURITY EVALUATION CRITERIA IN CERTAIN ENVIRONMENTS. National Computer Security Center, CSC-STD-003-85 25 June 1985. (Yellow book)

8. Government of Canada, CANADIAN TRUSTED COMPUTER PRODUCT EVALUATION CRITERIA, Canadian Systems Security Centre, Communications Security Establishment, Government of Canada, Version 2.ie, July 1991.

9. UK IT CESG, "UK IT Security Evaluation and Certification Scheme", undated pamphlet.

10. USAF. AIR FORCE TRUSTED CRITICAL COMPUTER SYSTEM EVALUATION CRITERIA (TCCSEC). U.S. Air Force HQ Electronic Security Command. San Antonio, TX 78234-5000. 25 June 1990. Draft

11. USAF. AIR FORCE TRUSTED EMBEDDED COMPUTER EVALUATION CRITERIA INTERPRETATION (TCCSECI). U.S. Air Force HQ Electronic Security Command. AFCSC/SRVC. San Antonio, TX 78234-5000. 25 June 1990. Draft

12. K.J. Biba. **"Integrity considerations for secure computer systems".** Technical Report MTR 3153, The Mitre Corporation, Bedford, MA, June 1975. Also available from USAF Electronic Systems Division, Bedford MA, as ESD-TR-76-372, April 1977.

13. D.E. Bell and L.J. LaPadula. Secure Computer Systems. Three volumes. Technical Report MTR-2547, The MITRE Corporation Bedford, MA, March-December 1973.

14. D.D. Clark and D.R. Wilson. **" A Comparison of Commercial and Military computer Security Policies".** In Proc. 1987 Symposium on Security and Privacy, pages 184-194, Oakland, CA April 1987. IEEE Computer Society.

15. T.A. Parker. **"Restating the Foundation of Information Security".** Proceedings 14th National Computer Security Conference, pages 480-493. October 1991

16. J.C. Nagengast. **"Defining a Security Architecture for the Next Century".** Journal of Electronic Defense, Jan 1992 Pages 51-53.

17. C. Cecere and W. Ruppert. **"A Framework for Systems Security".** Journal of Electronic Defense, January 1992, Pages 54-56.62.63.

18. J.D. Weiss, **"A System Security Engineering Process".** In Proc. 14th National Computer Security Conference, pages 572-581. October 1991.

19. NSA. Information System Security Products and Services. Available quarterly from U.S. Government printing office.

20. K.A. Siil. **"Experiences in Multi-Level Security on Distributed Architectures".** In Proc. 14th National Computer Security Conference, pages 205-214. October 1991.

21. R. O'Brien and C. Rogers. **"Developing Application, on LOCK".** In Proc. 14th National Computer Security Conference, pages 205-214. October 1991.

INTEGRITY AND ASSURANCE OF SERVICE PROTECTION IN A LARGE, MULTIPURPOSE, CRITICAL SYSTEM

Howard L. Johnson
Information Intelligence Sciences, Inc.
1903 So. Franklin St., Denver, Colorado 80210

Chuck Arvin and Earl Jenkinson
CTA Incorporated
7150 Campus Drive, Suite 100, Colorado Springs, CO 80918

Captain Bob Pierce
AF Cryptologic Support Center
Hq. AFIC, AFCSC/SR, Kelly AFB, TX 78243-5000

ABSTRACT

This paper is the third and last in a series to discuss goals and concepts of the "Air Force Trusted Critical Computer System Certification Criteria." The first paper described an approach to protect against the malicious logic threat for a DoD system. The second, building on the first, identified how a system with a critical mission uses strict resource allocation and time constrained response to help ensure mission success. This paper, building on the previous two, addresses integrity and assurance of service protection in large multi function systems, where only some of the functions are critical to National objectives or human safety. Results use and expand on principles of the Orange Book. A final section defines an approach to integrating the three security objectives: confidentiality, integrity, and service assurance.

BACKGROUND

[1] described an approach to protect against malicious logic for a DoD system. [2] identified how a system with a critical mission uses strict resource allocation and time constrained response to help ensure mission success. Key results, that form the basis for this paper, are presented in Appendix 1. The three paper series is a synopsis of requirements and issues from the Air Force Trusted Critical Computer System Certification Criteria (AFTCCSCC) [3]. The ideas were first presented in preliminary form in [4].

Application of the forms of protection functionality and assurance described by the TCSEC [5] are also necessary for integrity and assurance of service protection in DoD applications. However, additional mechanisms are also required. It is important to note that these additional mechanisms would be equally useful to augment Orange Book requirements to help defeat a malicious attack threat whose objective was to gain unauthorized access to sensitive/ classified information, while still assuring accomplishment of critical mission functions.

PROBLEM

The TCSEC does not adequately address attacks that implant malicious code. A confidentiality policy that allows upgrading of data, all but invites malicious code insertion and execution (e.g., by other cooperative malicious code) in often critical, Top Secret operations. However, for an attacker to release protected data requires exploitation of a flaw or a covert channel. In an integrity/denial of service attack, the attacker must find a way (e.g., a covert input channel) to insert and execute code. Once inside, no other flaw (analogous to the leakage path) is needed. Even the single level solution [2], where system data and functionality are critical, weakens as the system becomes large and complex, with exposure to many users. The reason is increased risk of covert data input channels and difficulty of assurance due to combinatoric effects.

Protection Domains

In Panama, the U.S. President was isolated in a crowded environment, using trusted individuals and special procedures. This is because his function is deemed (at least by the U.S.) as more critical than other people present; plus he is considered a high probability target. The TCSEC takes a somewhat analogous approach with the TCB. Security is critical and the TCB is vital to security protection. Personnel trust is a factor in determining division/class [6], which dictates TCB strength. Goals in building the TCB include isolating critical functionality and reducing complexity to minimize the probability of exploitable flaws. It is especially undesirable for a rogue program to become part of the TCB, because it could use the special privileges to compromise security.

Isolation by Criticality Level

For the same reasons given above, it makes sense to isolate and minimize functionality vital to accomplishing a critical mission. There can be levels of isolation based on gradations of criticality of functions and data to the mission objective, where part of that objective is critical or highly critical to National goals and/or human life. The functionality that eliminates the possibility of the unauthorized launching of a nuclear weapon might be more critical than assuring proper support of an authorized launching. Two independent critical functions, conventional weapon firing and operator safety, might be considered equally critical, but isolating these functions from each other can provide a higher integrity assurance for each. It makes no sense for highly critical functions to be exposed to additional risk inherent to programs, data, and users of other functions.

Data flow control can help minimize the possibility of malicious logic insertion or other unauthorized changes originating from outside or at a lower (less critical) level. This isolation is similar to that proposed in the Biba integrity model [7]. However,

isolation and restricted data flow are only a small part of the security protection, as can be seen from Appendix 1.

The need for a hierarchical protection also arises from fiscal considerations. Costs of special procedures, special background investigations, and special vaults and containers may only be justifiable valid for Top Secret data, with less expensive approaches used where data is not as sensitive. Analogously, more money should be allocated to system security for more critical functions than systems where the impact of loss would be less.

Partial Ordering

Applying the algebraic property of partial ordering also provides additional protection. If the transitive subproperty of partial ordering is not met, malicious code can compromise the policy, as illustrated in Figure 1. The hierarchical properties also provide a set of rules to remind programmers and users that to allow into a critical environment either data or programs that have not been assured to be malicious logic free -- is to make a terrible mistake.

CRITICALITY LEVELS AS A CONCEPT

Relationships and differences between levels of sensitivity, criticality, and integrity are important to our concept. (Note that in this paper, the term sensitivity is analogous with confidentiality.) Data security classification and mission criticality are not generally related (rather they are orthogonal requirements). It is desirable for both sensitive and critical functions to have high integrity, but integrity might be desired for other reasons in a critical function (e.g., accuracy, fidelity, or consistency). Criticality pertains to the desire for mission success. For critical functions, we minimize the number of personnel involved and take special precautions (e.g., background investigations) to ensure greater trust. The higher the criticality, the greater the required trust.

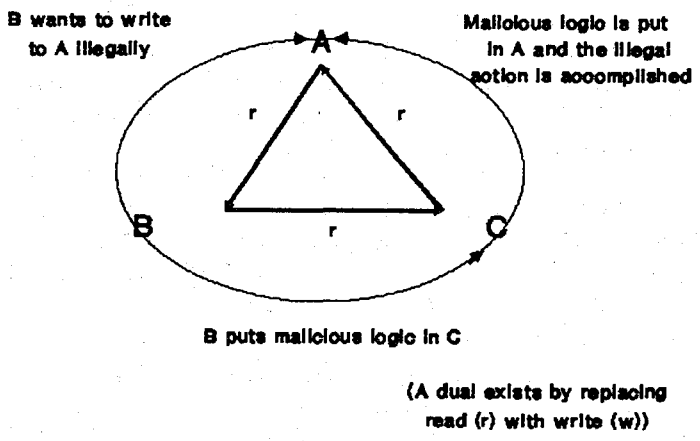


Figure 1 Non Lattice Malicious Threat Example

Security Policy

To briefly review the mandatory policy presented in the AFTCCSCC, three criticality levels are specified (Highly Critical, Critical, and Noncritical). The following is an excerpt from the flow policy (illustrated by Figure 2 from []).

Rule 1 (simple security property): A subject is allowed read access to an object only if the criticality level of the object dominates the criticality level of the subject.

Rule 2 (confinement property): A subject is allowed write access to an object only if the criticality level of the subject dominates the criticality level of the object.

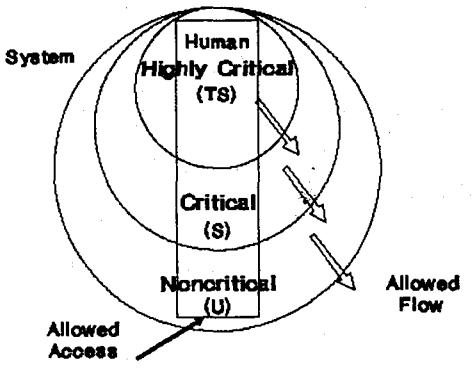


Figure 2 Criticality Levels, Data Flow, and Authorization

Rule 3 (execution rule): A subject is allowed execute access to a program only if the criticality level of the program dominates the criticality level of the subject.

Criticality level A is said to dominate criticality level B if the hierarchical criticality level is greater than or equal to that of B and the nonhierarchical categories of A include those of B as a subset. Highly Critical is greater than Critical and Critical is greater than Noncritical. To make implementation practical for the DoD, it is suggested that a person have a Top Secret Clearance to be allowed Highly Critical access and a person have a Secret Clearance to be allowed Critical access. (However, at the DAA's discretion, other criteria can be used.) A person with a particular access also has authorization for lower levels.

Treatment of Categories

In a "write," in criticality, the categories of the subject must be a subset of the categories of the object, so malicious code is not introduced into a new category. This supports the concepts of "need-to-modify" or "need-to- execute." In a "read," categories of the object must be a subset of the categories of the subject to not introduce malicious code from a new category. The term "dominates" can be interpreted as "is a subset of." (Sensitivity that supports the concept of "need-to-know" in which "dominates" can be interpreted as "is a superset of.") In sensitivity, association with a category is an added privilege. In criticality it is a restriction of privilege.

SURPRISING FEATURES OF CRITICALITY PROTECTION

Two topics emerge from this policy definition: how to deal operationally with exceptions and how the "execute policy" is implemented. These are addressed in the next two sections. Other peculiarities of criticality are also discussed.

Dealing with Required Policy Violations

Security policy models are simple abstractions and conservative statements of control, but as such, do not always allow operationally required data flow. This is also true in practical applications of TCSEC multilevel security. "Policy violations" in the context of this paper refer to intended design of mechanisms that allow information flow contrary to security policy, while still supporting the fundamental security objectives. If a procedure can determine, with an acceptable degree of certainty, that no malicious logic exists in a set of data (including programs), then the data may be upgraded to a higher criticality level. Code and data can exist redundantly at different levels as in sensitivity. Data temporarily loaned can be proven unmodified (through modification detection mechanisms). Security guards should be used where policy violations must occur.

Execute Access

From a definition in the NSA Discretionary Access Control document [8] "Execute allows a subject to run the object as an executable file. On some systems, execute access requires read access." From the policy stated previously, a lower criticality level is not permitted to execute a program at the higher criticality level. (If allowed, this would permit the second half of a "pass-code-and-run-it" attack). Since a lower level has read capability, it could read a program from a higher level and execute it, but only at its own level. A higher criticality level is allowed to execute a lower criticality program, however, the higher criticality has write, but not read, privileges. The result is that a "write" down or trusted communication with the TCB must be associated with an authorized "execute" across criticality levels.

Risk Management Concept of "Exposure"

The sensitivity risk index is defined in [6] as a function of maximum data level (e.g., sensitivity security classification) and lowest trust level (e.g., security clearance). However, in criticality, risk is additionally a function of the lowest criticality level of data present in the system. Therefore, exposure can be defined as the difference between the maximum criticality level and the minimum of lowest trust level and lowest criticality level. Once data or functionality has been exposed to a lower level of criticality it must be labelled at the lower level until some process can ensure the nonexistence of malicious logic and certifies its safe use again at the higher level.

Applicability of Mode

Ideas of "dedicated mode" and "system high mode" introduced in [6] take on unwanted characteristics when applied to criticality. Automatic upgrade of data increases risk significantly. Automatic downgrade lessens the protection of critical functionality. The best strategy is multilevel isolation by "true" criticality level.

Protection Granularity

In integrity and service assurance protection, assignment of "criticality classification" is not simply at the data item level as it is in sensitivity, but is more fundamentally related to the different hardware, firmware, software, and data elements depended on to meet critical mission objectives. Criticality protection granularity must be consistent with (and adaptive to) the protection needs of these critical elements. A primary concern is the insertion of a malicious code string. Code can be inserted a little-at-a-time, analogous to the leaking of classified, though the problem of reassembly creates more complexity for the attacker. In conclusion, the granularity of criticality protection is similar to sensitivity, but is determined by completely different factors.

Semantics and Syntax

Protection of sensitive data is normally a semantic issue, and only occasionally a syntax issue. Classified information can usually be conveyed in many ways (e.g., verbally or graphically), where the human must determine the context and filter information according to the situation. Malicious logic, unauthorized execution or any means to modify data, are more likely to be a syntax problem (certain spoofs are exceptions). Malicious logic must be syntactically precise to be an effective attack.

Labels and Exportation

Criticality marking is analogous to sensitivity marking for stored electronic information. However, printed information requires no criticality marking, since malicious logic generally cannot be passed unaware through a human back to electronic form, retaining the detail required. If the TCB exports an object to a multilevel I/O device that does not accept data in machine readable form, the label requirement can be dropped. Examples are devices driven by computer generated control data. (Future applications using print output and scan/recognition input might be an exception.) The user should be continually aware of the criticality level of operations and the task being run to prevent inadvertent compromise (e.g., loading uncertain input data). This can be accomplished by visual headers or other means.

COMBINING CRITICALITY AND SENSITIVITY

Nothing in the TCSEC or its application is altered by the AFTCCSCC. For a system, either the TCSEC or the AFTCCSCC may be applied, or both may be applied. A division/class assignment from the TCSEC is unrelated to the division/class from the AFTCCSCC. The stronger of shared mechanisms (e.g., identification/authentication) will also satisfy the weaker requirements. Integrity and confidentiality (not considering denial of service) have been merged in many systems [9]. Merging sensitivity and criticality is analogous. Neither takes precedence; both must be satisfied.

Figure 3 shows the division/classes of the AFTCCSCC. Given a class (say F1) and a TCSEC class (say B2), the requirements can be laid side-by-side in each policy area (e.g., identification/ authentication and audit). The requirements have been written so they can be easily merged. First, the policy models must be merged into a single model. (A graphical presentation of a combined lattice is shown in Figure 4.) The context of the requirement indicates whether: 1) both requirements need to be supported separately, 2) they need to be further broken down, 3) one takes precedence over the other because it is stronger, or 4) they can be combined into a single requirement. If they are independent, it must be ensured that they do not conflict with each other or any other requirements, and that they fully specify the capability. Conflicts must be resolved. If requirements are incomplete, they must be augmented.

Criticality Division/Class	Protection
H	Same as TCSEC D
G	Single Level
G1	Almost the same as TCSEC C1
G2	Protects against malicious logic
G3	Supports Critical operations
F	Multilevel (Labels)
F1	Critical and Highly Critical
F2	Critical and Non Critical
F3	No clearance and Critical
E(E1)	Formal methods (no clearance and Highly Critical)

Figure 3 AFTCCSCC Division/Class

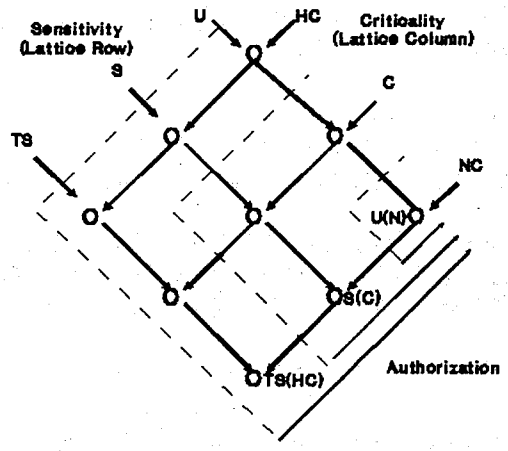


Figure 4 Flow and Authorization Lattice

Sensitivity and Criticality as Mechanism Protection Requirements

A classified mission may not be critical. Nevertheless, mechanism integrity and service concerns exist. A critical mission may not deal with classified data. Even so, there are sensitivity aspects to criticality mechanisms (e.g., passwords, keys, or mechanism knowledge). Highly Critical mechanism sensitivity aspects should be protected at Top Secret and Critical mechanism sensitivity aspects should be protected at Secret (or some DAA authorized equivalent).

Retrofitting TCSEC Protected Systems

There is no experience, but it is felt that a B2 protected computer system can be retrofit with G2 through F2, a B3 can be retrofit at F3 and below, and an A1 can be retrofit with any criticality division/class. The effort and cost of retrofit will depend on the individual system. A C2 computer system can probably only be retrofit with a subset of G2 or G3 requirements. The primary problem is the required strength of the TCB and the capability of the reference monitor.

SUMMARY

It has been argued that partially ordered levels with categories are important to integrity and assurance of service protection for large systems with some critical requirements. It is shown how TCSEC and AFTCCSCC protection objectives can be simultaneously pursued. A summary of the key requirements discussed here are presented in Appendix 2. It is hoped that the requirements of both Appendix 1 and 2 will be seriously considered in building future National and Federal security criteria.

BIBLIOGRAPHY

[1] Johnson, H.L., C. Arvin, E. Jenkinson, B. Pierce, "A Proposed Approach for the Air Force to Deal with the Malicious Logic Threat," Proceedings 14th National Computer Security Conference, NIST and NCSC, October 1-4, 1991, pp. 137-146

[2] Johnson, H.L., C. Arvin, E. Jenkinson, B. Pierce, "Proposed Security for Critical Air Force Missions," Proceedings 7th Annual Security Applications Conference, December 2-6, 1991, pp. 209-217

[3] Air Force Trusted Critical Computer System Certification Criteria, Air Force Special Security Manual 5029, Air Force Cryptologic Support Center, August 14, 1991

[4] Johnson, H.L, "Security Protection Based on Mission Criticality, Proceedings Fourth Aerospace Computer Security Applications Conference, IEEE, December 12-16, 1988, pp. 228-232

[5] DoD 5200.28-STD, "Trusted Computer System Evaluation Criteria," December, 1985

[6] DoDD 5200.28, "Security Requirements for Automated Information Systems (AISs)," March 21, 1988

[7] Biba, K.J., "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA, April, 1977

[8] NCSC-TG-003, A Guide to Understanding Discretionary Access Control, NCSC, 30 September 1987

[9] Lipner, S.B., "Non-Discretionary Controls for Commercial Applications," Proceedings 1982 IEEE Symposium on Security and Privacy, 26-28 April 1982, pp. 2-10

APPENDIX 1

Protect against the malicious logic threat for a DoD system (G2)

1. Provide a TCB safe from malicious logic attacks.
2. Provide a trusted path between users and the operating system with mechanisms to avoid replay and spoofing attacks.

3. A reference monitor-like function must disallow, by process and by access mode, access to objects not intended by the design
4. Selectively use encryption or other coding schemes to isolate data/programs, to detect modification of data/programs, and to authenticate origin, time, and identity
5. Assure there are no uncontrolled paths to insert and/or execute malicious code
6. Provide for near-real-time detection of select operational abnormalities that suggest a malicious intrusion
7. Specify that programs report internal faults potentially attributable to a malicious attack
8. Require a resource scheduling policy, violation of which could be considered a potential denial of service attack
9. Search off-line media to identify known malicious or suspicious logic
10. Provide a fault source identification "expert" to assist in determining if a fault is of malicious origin
11. Emphasize safe development and life-cycle configuration management of programs and data
12. Augment verification and test to identify existence of malicious logic or the presence of malicious intent
13. Perform penetration testing to determine strength against malicious attack.

Protect critical missions against loss of integrity and denial of service attacks (G3)

1. Use background investigations to establish user trust
2. Define mission accomplishment under all conditions
3. Determine time available to identify and fix problems
4. Develop and maintain a time budget and place it under configuration control
5. Use concurrency and graceful degradation to meet time budget
6. Identify appropriate response for each detected abnormal condition
7. Provide resource allocation policy for normal, peak, and degraded conditions
8. Detection criteria must be alterable during attack and after response feedback
9. Provide trusted recovery, diagnosis, and repair in response to detected problems
10. Time vulnerabilities should not be revealed through documents or operations (e.g., traffic) analysis
11. Recovery should utilize hardware, information, and software redundancy
12. Design shall avoid common vulnerabilities in redundant elements
13. Use problem isolation and containment
14. Use diagnostics, fault source experts, and person-in-the-loop
15. Repair shall control, eliminate, or bypass the intruder and/or his code and replace faulty program or data
16. Testing shall consider system resilience

APPENDIX 2

Integrity and assurance of service protection in large multi function systems, where only some of the functions are critical to National objectives or human safety. (F1 through E1)

1. Isolate and minimize functionality vital to critical missions
2. Enforce least privilege in criticality
3. Mandatory criticality shall support partial ordering, a simple security property, a confinement property, and an execute policy that prevents malicious code insertion and execution
4. Criticality upgrade requires certification, modification detection, and a guard
5. Criticality exposure is defined as the difference between the maximum criticality level and the minimum of lowest trust level and lowest criticality level
6. Once data or functionality has been exposed at a lower level of criticality it must be labelled at the lower level
7. In a "write," categories of the subject must be a subset of the categories of the object
8. In a "read," categories of the object must be a subset of the categories of the subject
9. "Dedicated" and "system high" modes increase criticality risk
10. Criticality is dynamically assigned to hardware, firmware, software, and data depended on to meet critical mission objectives
11. Labeling is based on syntax
12. Printed information requires no criticality marking
13. Labeling is only required machine readable data
14. Highly Critical mechanism sensitivity should be protected at the Top Secret level and Critical mechanism criticality aspects should be protected at the Secret level
15. The user should be made aware of system and task criticality

Internetwork Security Monitor: An Intrusion-Detection System for Large-Scale Networks

L.T. Heberlein, B. Mukherjee, K.N. Levitt

Computer Security Laboratory
Division of Computer Science
University of California
Davis, Ca. 95616

ABSTRACT

The model for an Internetwork Security Monitor (ISM) is presented. The objective of the model is to significantly improve our capability to detect and react to intrusions into an arbitrary wide-area network (WAN) (e.g., the Internet) through a distributed intrusion-detection and analysis system. The system will monitor the various component networks of the internetwork and bring potentially intrusive behavior to the attention of the local-network security managers. The model primarily extends the DIDS and NSM intrusion-detection systems and takes advantage of, but does not require, cooperative host monitoring. This design will provide the first intrusion-detection system that aggregates information from different monitors over wide-area networks and will be deployable at different sites with widely different operating environments and security requirements.

1. INTRODUCTION

The prevalence and ease of use of networking to provide remote access to resources has brought with it a set of previously unanticipated problems. Network managers worldwide are extremely concerned with the problem of network intrusions, which are unwanted or unauthorized use of the network to gain access to (and sometimes modify) both network and computing resources. These intrusions have often appeared in the popular news media and have been a serious impediment to many organizations obtaining network connection.

What do we mean by network intrusion? For our purposes here, we consider a network intrusion to be any unwanted or unauthorized actions being taken across the network that affect remote resources. These actions include those of the "Wily Hacker" [Sto89]—where the intruder aims to gain unauthorized access to information on a number of computers on the network—unauthorized remote modifications of router tables in an Internet, and attempts to deny use of the network to authorized users.

Examples of network intrusions that concern operators include:

- unauthorized modifications of system files that permit unauthorized access to either system or user information
- unauthorized access to user file space
- unauthorized modifications of user files/information
- unauthorized modifications of tables or other system information in network components
- unauthorized use of computing resources (perhaps through the creation of unauthorized accounts or through the unauthorized use of existing accounts.)

Intrusion Detection Systems (IDS) attempt to detect the presence of such attacks. Early IDS were designed around the analysis of a single host's audit trail. Their examples are SRI's early model of the Intrusion Detection Expert System (IDES) [Den87], National Security

Agency's MIDAS, Haystack Laboratories' Haystack System [Sma88], Los Alamos National Laboratory's Wisdom & Sense (W&S) [Vac89], and AT&T's ComputerWatch [Dow90]. However, with the proliferation of computer networks, many of these IDS began to apply their host based techniques to small networks of computers. Their examples include SRI's IDES [Lun90] and the Distributed Intrusion Detection System (DIDS) [Sna91].

Unfortunately, even with the extension of IDS into small networks of computers, because the networks are often interconnected, an IDS's ability to detect intrusive activity and to determine the party responsible for such activity is limited. Intrusion detection is an internetwork problem. Often intrusions or attacks affect more than a single network, and detection may require exploitation of data from multiple networks or computers.

To address these limitations, we designed a model, called the Internetwork Security Monitor (ISM), to perform intrusion detection in a highly interconnected wide-area network. Specifically, our ISM design requires the development of a hierarchical internetwork monitor as an extension of ongoing work in distributed-intrusion detection. In extending the LAN monitoring capabilities into an internetwork environment, we are exploring the feasibility of different design alternatives for distributed-network traffic monitoring and analysis, including the following hierarchical architecture. Under this architecture, independent monitors are placed at various locations over an internetworked environment. These monitors exchange and share information (including those on hypothesized attacks) to detect possible security breaches. Subnetworks, in turn, exchange information among one another to detect inter-subnetwork attacks.

The scenarios in Section 2 motivate our work by describing the type of behavior that our internetworked security monitor model is designed to detect and analyze. Section 3 presents an overview of the ISM components. Section 4 discusses how complete accountability can be attained in a networked environment. Section 5 presents the ISM model as an extension of current work being done. Finally, Section 6 provides some concluding remarks.

2. SCENARIOS

Because the Internet is distributed, the evidence needed to detect an intrusion may also be distributed across the Internet. For example, suppose an intruder systematically attacks hosts at a particular organization (site A) until he successfully penetrates a host. This attack method, called the doorknob attack, may be successfully detected at site A. However, once the intruder has acquired a foothold on a computer at site A, he may notice a *.rhosts* file in a user's home directory, which indicates that the user trusts logins from computers at a second organization (site B). Hoping the trust is mutual (i.e., the user has *.rhosts* files in his accounts at site B for logins from site A), the intruder could masquerade as this user at site A and successfully log into a computer at site B.

Since this login would be between two machines which do occasionally exchange logins, and since no vulnerabilities other than trust would be exploited, site B's intrusion-detection system would be unable to discern this login as an intrusion.

A second scenario is based on an actual attack detected and analyzed by the Network Security Monitor (NSM) [Heb91]. This attack also begins as a doorknob attack. The intruder, attacking across the Internet from site A, attempts to penetrate over sixty computers before eventually finding one with the default (and flawed) system configuration in place. Once the intruder penetrates this host, the attacker quickly inserts a Trojan login program, and prior to the intruder exiting the penetrated machine, a login from site B successfully exploits the newly installed Trojan login program. The intruder from site B remains logged in for several hours exploiting various bugs in systems as well as the trust between the organization's machines.

Investigation the next day shows that neither of the hosts from sites A or B were the root of the attacks. Their systems had also been attacked and merely used as launch pads to attack the computers.

The next night, the intruder penetrates the machines from a third organization (site C). Detecting the penetration, we examined the host at site C as the intruder, but the host at site C, obviously subverted, reports that no one is logged on. Our trail has gone cold again.

From these and other incidents, we are convinced that we have very little chance of catching intruders originating outside our organization. With current intrusion-detection techniques, we can detect many intrusions into our systems, but attacks from outside are relatively difficult to dissect.

3. ARCHITECTURE OVERVIEW

The ISM model extends research and development efforts already existing in the field of intrusion detection. Primarily, the ISM extends the Distributed Intrusion Detection System (DIDS) (see [Sna91]) into arbitrarily wide networks. Multiple DIDS-like monitors, called ISM domain monitors, communicating through well-defined protocols form the core of the distributed ISM. In addition to the monitors themselves, Security Domain Name Servers (SDNS), based on the Domain Name Server (DNS) model, provide a mechanism for the ISMs to locate each other across the Internet. Finally, security workbenches allow network managers to logon to their local ISM domain monitor to examine the results of the monitor's analysis, query further into possible intrusions, exchange information with other network security managers, and administer various security tools such as Security Profile Inspector (SPI) or Computer Oracle Password Security system (COPS). Although all three major components—ISM domain monitors, Security Domain Name Servers, and security workbenches—comprise the ISM model, this paper focuses on the ISM domain monitors.

4. ACCOUNTABILITY

One of the most fundamental and critical capabilities in a computer system security is establishing accountability for actions performed by individuals. A combination of authentication and auditing mechanisms residing in the operating system usually provides this accountability. In such systems, the user identifies and verifies himself (via a password) to the authentication mechanism, and the auditing mechanism keeps account of the activities performed by that authenticated user.

Unfortunately, the accountability can be lost when the user crosses operating system boundaries (e.g., logging into another host across the network). Although the user will be re-authenticated by the new machine (either with a new password or by trusting the authentication of the first host), the accounting of the user's activities will be distributed across the audit trails of multiple hosts. If users are restricted from changing their identification as they move across systems, and if the audit timing can be synchronized across auditing mechanisms, accountability can be achieved; however, such restrictiveness is not attainable in many environments.

4.1 NETWORK IDENTIFIER

The Distributed Intrusion Detection System (DIDS) was designed in part to achieve an accountability across a network of heterogeneous systems. When a user initially signs on to one of the components of the network, that user is assigned a Network Identifier (NID). As the user moves across the network of computers, all activity performed by that user on any host is mapped to the NID. Therefore, accountability across the network is established.

To account for a user's activities across the network, DIDS, working with the established auditing mechanism on each host, creates a map between a user's UID for a session and an NID. A user's activities across a network can be accounted for by extracting the activities from each host associated with a UID which maps to the same NID.

DIDS creates the map between the UID on a host and the NID by tracking a user's movement across the network and exploiting transitivity. For example, if user A on host1 performs a remote login across the network to host2 as user B, DIDS tracks A@host1 to B@host2, and the instance B@host2 is mapped to the same NID as A@host1. If the user performs a second remote login from host2 to host3 as user C, we can use the simple rules

NID(C@host3) = NID(B@host2)
and NID(B@host2) = NID(A@host1)

to conclude that the NID for C@host3 is the same as the NID for A@host1.

The tracking between users and hosts is performed by treating a network connection as a shared resource and determining which users are accessing that resource. For example, if a user creates a remote login session (called session2), on host2, DIDS first identifies the host-to-host connection (called net-rsrc) responsible for the session and binds the information as the pair <net-rsrc, session2@host2>. DIDS then determines which session on host1 meets the requirements <net-rsrc, ?@host1>, and tracking is achieved (see Figure 1).

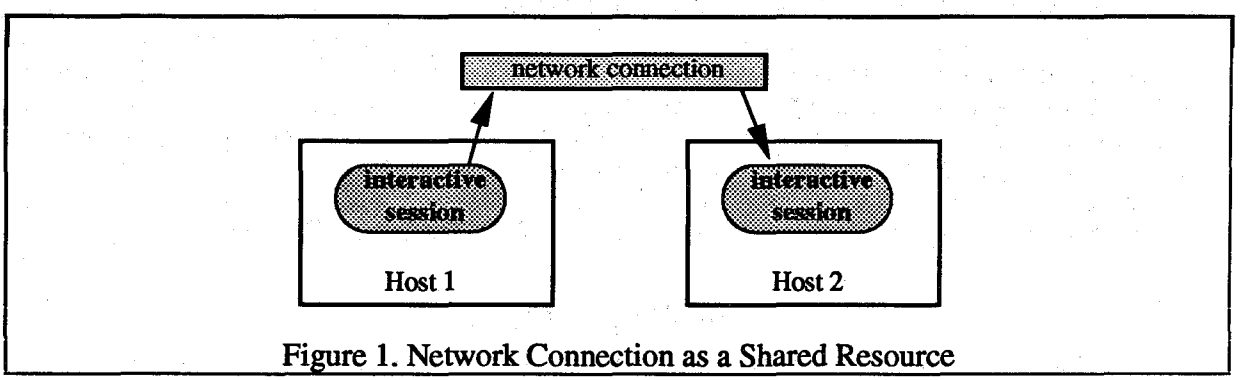


Figure 1. Network Connection as a Shared Resource

Unfortunately, in many environments, not all computers on the network support a host monitor which provides either the accountability for that particular host or the information required to track users across the network. In such environments, security and accountability can be increased by using a network monitor such as the Network Security Monitor (NSM).

4.2 TRACKING USERS THROUGH HOSTS WITHOUT MONITORS

The NSM, initially designed to detect intrusive activity across a local-area network (LAN), already augments DIDS' analysis capability by scrutinizing network activity into hosts which do not support host monitors; therefore, all hosts in the DIDS domain can be monitored to a certain level for the presence of intrusive activity. However, the current DIDS system cannot perform the NID tracking when a user passes through an unmonitored host. The following example illustrates the problem.

Suppose the DIDS-monitored domain consists of three hosts, two of which support host monitors (hosts one and three) and one which does not (host two). In this domain, a person initially signing onto host1 and then performing a remote login to host3 will have all of his activities mapped to a single NID, so DIDS maintains complete accountability. However, if the person first performs a remote login to host2 (the unmonitored host) and then performs a second login from host2 to host3, the user's activities on host3 will not be mapped to the same NID as his activities on host1, so DIDS loses complete accountability (see Figure 2).

Our challenge has, therefore, expanded to obtaining complete accountability across all monitored hosts by mapping a user's activities on all these hosts to the same NID even if the

user temporarily leaves the domain of monitored hosts. Fortunately, the mapping of a user's activities can still be obtained by tracking the user by connections, even through unmonitored hosts, if we simply expand our notion of a network connection.

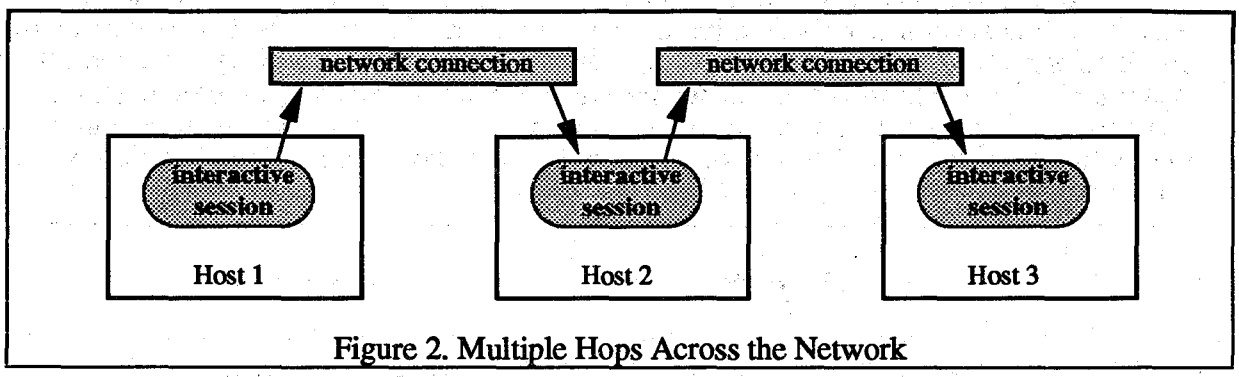


Figure 2. Multiple Hops Across the Network

4.2.1 EXTENDED CONNECTIONS

The previously described DIDS algorithm attains a map of a UID to an NID by tracking the user's login session back to the original login session. DIDS performs its tracking by determining the ownerships of a shared resource, namely a network connection, and recursively applies the procedure until DIDS reaches the original login session. However, the recursion fails when one of the hosts involved is an unmonitored host. The following conceptual extension to network connections allows us to continue the recursive algorithm through unmonitored hosts.

In the previous example, a user on host1 performs a remote login to host2 and then performs a second remote login to host3. Figure 2 presents a logical view of the user's actions. Using some I/O device (e.g., a terminal) connected to the session on host1, the individual can perform actions on host3 and view the results as if he were connected directly to the session on host3. This "virtual," direct connection occurs because the session on host2 is acting as a repeater. Thus, all commands and results are passed through the session on host2 unaltered. By exploiting this invariance, we can view the two network connections in Figure 2 as components of a single "extended" connection between the session on host1 and the session on host3. Now when the DIDS recursive algorithm used to track users encounters an unmonitored host, the algorithm can bypass the host by exploiting the extended connection, if one exists, and continue the algorithm at the next monitored host.

Formally, we define an extended connection as a set of network connections used to transport data and control between two sessions. Figure 3 shows an extended connection in relationship to data, host-to-host protocols, point-to-point protocols, intermediate sessions, and routers. As Figure 3 shows, only the data (e.g., control information sent from host1 to host3) remains invariant across the various network components. By exploiting this invariance, via a method we call thumbprinting, the NSM maps the various host-to-host connections to the same extended connection.

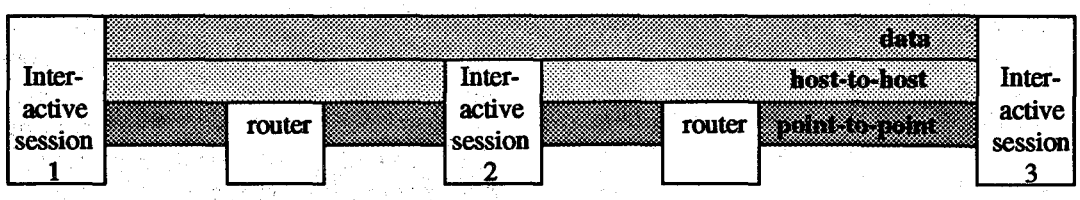


Figure 3. Extended Network Connection

4.2.2 THUMBPRINTS

The NSM maps host-to-host connections to an extended connection by assigning to each host-to-host connection a thumbprint representing the data flow for that connection for a specified period of time and then comparing the thumbprints for the various connections. If the thumbprints for two host-to-host connections match (within a measure of tolerance), they are mapped to the same extended connection. Thumbprinting even works when the number of intermediate, unmonitored hosts, between two host-to-host connections is unknown (see Figure 4).

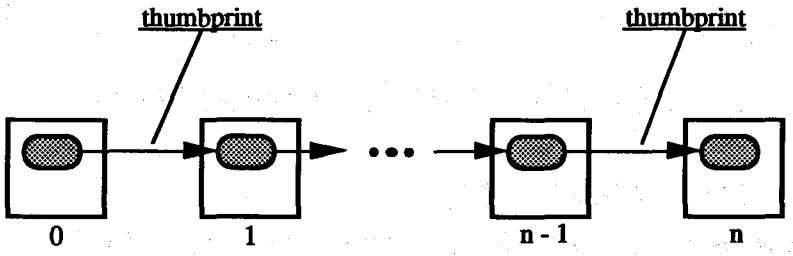


Figure 4. Tracking a User Across Unmonitored Hosts

We formally define a thumbprint for a host-to-host connection as a vector, $\mathbf{X} = \langle x_1, x_2, ..., x_n \rangle$, where each x_i is a counter for the occurrence of some attribute in the data. Two thumbprints, **X** and **Y**, are compared for similarity by determining the distance between the two vectors, $| \mathbf{X} - \mathbf{Y} |$. The certainty that the two connections which created the thumbprints are actually part of the same extended connection is inversely related to this magnitude.

The mapping of data in a host-to-host connection to a thumbprint vector, although extremely important, is not necessarily uniquely defined, and we are experimenting with various implementations. The implementations are driven by several goals described in Table 1.

Table 1. Thumbprint Implementation Goals

Resolution	The primary purpose of the thumbprint is to correctly recognize that two host-to-host connections are part of the same extended connection.
Semantic Free	As will be seen later, thumbprinting will be used in an open environment where privacy is an issue; therefore, the thumbprint, while being able to represent the connection, should not reveal the contents of the connection data.
Efficiency	The calculation of each x_i as well as the calculation of $| \mathbf{X} - \mathbf{Y} |$ must be efficient in order to allow for the thumbprinting of thousands of simultaneous connections and their comparison in real time.

Up to this point, we have argued for the need for accountability in computer systems, and we have shown that for complete accountability across operating system boundaries, we need to be able to track users across the boundaries. DIDS has proven that tracking can be performed in a small network of monitored hosts, and we have described an extension to DIDS allowing us to track users across an unknown number of unmonitored hosts. We now present an architecture based on NSM and DIDS which provides for intrusion detection and accountability in large-scale interconnected networks (e.g., the Internet).

5 ISM

The ISM model links together security systems monitoring particular domains (e.g., a DIDS-monitored domain) via standard information exchange protocols such as the Common Management Information Protocol (CMIP) to create a large-scale, highly distributed intrusion-detection system. The model is flexible in that different security domains can choose their own level of security analysis, from virtually none to complete transaction-to-transaction analysis, as long as they provide a minimum set of functionality described below. Finally, the model is hierarchical and supports the current network management structure by hiding a site's internal security structure from outsiders.

5.1 ISM PEER-LEVEL COMMUNICATION

An ISM is responsible for a specific set of hosts. When a user initiates a connection from a host in one ISM domain to a host in a second ISM domain, the ISMs may exchange information to allow a more accurate analysis of the security state of their own domains. At a minimum, an ISM must be able to identify the source (local or external to the domain) for connections leaving its domain. If the user initiating the connection originated inside the ISM domain, the ISM need only respond that the connection began internally and not reveal the actual origin of the user. If the connection originated outside the ISM domain (e.g., the user merely passed through the domain), the ISM must respond with the host-to-host connection definition of the connection entering the domain. This minimum capability of an ISM prevents an intruder from exploiting the domain in an attempt to disguise his origin. The protocol to support this functionality is presented below:

- GET TIME <time>
- GET CONNECTION TCP/IP-DEF <def> TIME <time>
- GET ORIGIN CONN-ID <id>

The first request allows an ISM to synchronize its clock to the remote ISM. An alternate, and preferred method is to assume all monitors are running under a time protocol (e.g., the network time protocol, NTP). The second request (with the time given in the remote ISM's time frame) returns an identifier, which can be used to make further requests. The third request, fulfilling the minimum requirement for an ISM, returns the origin of the user (relative to the local ISM) as either local to the domain or external (including the TCP/IP-DEF).

Other functionality for an ISM, while helpful but not required, includes the ability to analyze the activity within the domain for intrusive activity. Access to this analysis by external ISMs are made by the following requests:

- GET ANALYSIS CONN-ID <id>
- GET ANALYSIS HOST-ID <host-address>
- GET ANALYSIS SERVICE <service-name>
- GET ANALYSIS VULNERABILITY <vulnerability-id>

The first request returns a value between 0 and 100, which indicates whether or not the ISM believes that the user owning the connection given by <id> is behaving intrusively. The

second request also returns a value between 0 and 100, indicating whether or not the ISM believes that the host is associated with intrusive activity. The host does not necessarily have to be within the ISM's domain. For example, if one ISM believes it is receiving a number of possibly intrusive connections from a particular host, it can query other ISMs as to whether they believe the host has a hostile user on it. The third request returns a value between 0 and 100 indicating the ISM's belief that service <service-name> is being used in an unusual and intrusive manner (e.g., when the Internet worm exploited a hole in the mail service). The last request returns a value between 0 and 100 indicating the ISM's belief that a particular vulnerability has recently been exploited. To perform this, the ISM must have a catalog of known vulnerabilities and signatures to detect their [attempted] exploitation. Due to the sensitive nature of vulnerabilities, some ISMs (e.g., those at government sites) may have a more complete listing than other ISMs (e.g., those at universities).

As an example, Figure 5 shows three ISM domains in which a single user accesses hosts in all three domains. ISM1 is able to observe all hosts within its domain; however, the hosts inside the second and third domains are hidden from ISM1's view. When a user connects to ISM1's domain, ISM1 queries ISM2 for the source of the connection, and ISM2 responds that the source is external and supplies the TCP/IP definition of the connection to ISM1. ISM1 can use this definition to query ISM3 and determine whether the source of the connection into ISM1 is somewhere inside of ISM3.

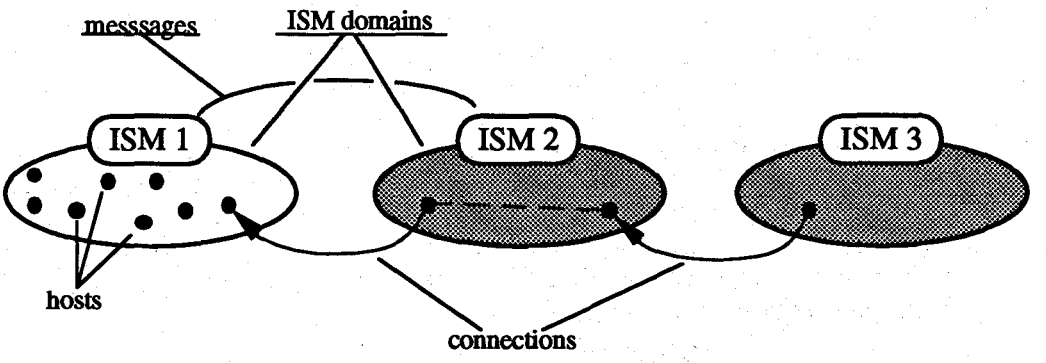


Figure 5. ISM Security Domains

5.2 ISM HIERARCHICAL COMMUNICATIONS

The ISM model also allows ISMs to be grouped hierarchically. For example, ISM1' may monitor a domain which is divided into three sub-domains, each with its own ISM sub-monitors. This hierarchical structure provides two major benefits. First, because the ISM1' domain can look into its sub-domains, it can aggregate a user's activities across these sub-domains. This functionality is provided by additional requests which can only be made by a direct parent ISM. These requests, however, can only be answered if the ISM sub-domain monitors support full tracking and accountability. The protocol to support these request are:

- GET NID CONN-ID <id>
- GET PATH NID <nid>
- GET VECTOR NID <nid>

The first request returns the NID associated with a given connection ID, and the NID can be used as a key to request further information. The second request returns an NID trace showing a user's movement throughout the domain. The third request returns a NID vector—a list of counts representing a user's activities in different categories (e.g., number of files opened or the number of times a specific command has been executed)—for the user in that

sub-domain. If a user's activities crosses several sub-domains, the parent ISM can trace all of the user's activities by requesting the paths and vectors of the user across all the sub-domains he crosses.

The second benefit of the ISM's hierarchical architecture is that internal structure can be hidden from outsiders. An individual site (e.g., a university or government research facility) may contain only a single ISM monitor (e.g., monitoring all traffic in and out of the site), or it may contain many sub-domains, each with its own ISM, divided along department lines. Whatever the structure, external sites can only view the site as a single ISM. The following example illustrates this ISM encapsulation, or information hiding.

Site A is composed of three ISM sub-domains, and site B is composed of two sub-domains. When a host in site A's third domain connects to a host in site B's first domain, site B's first domain cannot "see" site A's domain hierarchy, so it must send all queries to site A's parent ISM. Likewise, if site A's third domain queries site B for an analysis of the connection, the domain must send the query to site B's parent ISM. Importantly, to protect site A's internal structure, site A's third domain monitor performs its query through site A's parent ISM. Otherwise, a user at site B could determine site A's internal structure by "probing" site A and observing which internal ISMs respond to which probes. Meanwhile, site A's internal ISM domain monitors may continue to query each other locally (see Figure 6).

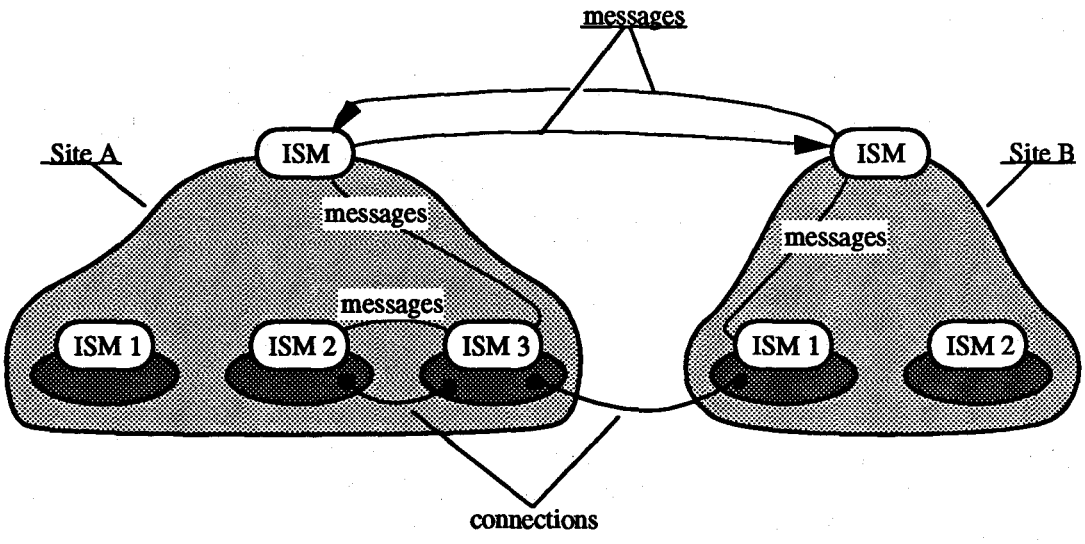


Figure 6. Security Domain Hierarchy

6. CONCLUSIONS AND FUTURE RESEARCH

Wide-area networks (e.g., the Internet) have grown to be large and complex, consisting of several thousands of networks (both wide-area and local-area) and managed by a comparably large number of organizations. Providing for coordinated network management in such an environment is a major task—one requiring advanced technologies that utilize the network and computing tools to assist in the management process. Nowhere does this issue show up more than in the area of network security. Because the Internet is distributed, evidence to identify and analyze an intrusion can be distributed over multiple sites on the Internet. Network managers at each site on the Internet must be provided with tools to analyze the evidence of an intrusion at the site and with tools to communicate their evidence and analysis with other managers so that the intrusion can be understood. The proposed ISM

design focuses on providing a distributed, intelligent, decision-support system for network managers that would partially automate the detection of intrusions into the Internet.

From an architectural point of view, the proposed ISM will make an important contribution towards providing security and management of the Internet; it will enable various subnets in the Internet to communicate with one another and to coordinate information in detecting potential attacks. As the Internet becomes larger, this decentralized architecture will avoid an information-flow bottleneck at the central processing node (funnelling point), which would occur under a centralized architecture.

Our future work includes using the NSM as a testbed to analyze various methods of thumbprinting. Not only are we analyzing methods with respect to resolution, efficiency, and semantic content (see Table 1), but we are exploring the possibility of mapping one thumbprint format into a second. For example, a government site may place a greater emphasis on high resolution than a university site, so they would be using two different thumbprint formats. However, if a mapping could be made from the high resolution thumbprint to the low resolution thumbprint, comparisons and tracking could still be performed.

Other future work includes testing, refining, and extending the protocols described here. As we move from design and testing to full implementation, we will probably find flaws in our initial design.

Finally, we are investigating attacks against time protocols, which can in turn subvert the effectiveness of thumbprinting.

References

[Den87] D.E. Denning, "An Intrusion Detection Model," *IEEE Trans. on Software Engineering*, vol. SE-13, no. 2, pp. 222-232, Feb. 1987.

[Den90] P.J. Denning, ed. *Computers Under Attack: Intruders, Worms, and Viruses.* New York: ACM Press, 1990.

[Dow90] C. Dowell and P. Ramstedt, "The COMPUTERWATCH Data Reduction Tool," *Proc. 13th National Computer Security Conference*, pp. 99-108, Washington, D.C., Oct. 1990.

[Heb91] L.T. Heberlein, B. Mukherjee, K.N. Levitt, D. Mansur., "Towards Detecting Intrusions in a Networked Environment," *Proc. 14th Department of Energy Computer Security Group Conference*, May 1991.

[Lun90] T.F. Lunt, et al., "A Real Time Intrusion Detection Expert System (IDES)," Interim Progress Report, Project 6784, SRI International, May 1990.

[Sma88] S.E. Smaha, "Haystack: An Intrusion Detection System," *Proc. IEEE Fourth Aerospace Computer Security Applications Conference*, Orlando, FL, Dec. 1988.

[Sna91] S.R. Snapp, J. Brentano, G.V. Dias, T.L. Goan, L.T. Heberlein, C. Ho, K.N. Levitt, B. Mukherjee, S.E. Smaha, T. Grance, D.M. Teal, D.L. Mansur, "DIDS (Distributed Intrusion Detection System)—Motivation, Architecture, and an Early Prototype," to be published in *Proc. 14th National Computer Security Conference*, Oct. 1991.

[Sto89] C. Stoll, *The Cuckoo's Egg*, Doubleday, 1989.

[Vac89] H.S. Vaccaro and G.E. Liepins, "Detection of Anomalous Computer Session Activity," *Proc, 1990 Symposium on Research in Security and Privacy,* pp. 280-289, Oakland, CA, May 1989.

Intrusion And Anomaly Detection: ISOA Update

J.R. Winkler and J.C. Landry

PRC, Inc.
MS:5S3
1500 PRC Dr.
McLean, VA 22102

(703) 556-1108
winkler_vic@po.gis.prc.com

Abstract

This paper presents an overview of the current status of the Information Security Officer's Assistant (ISOA) intrusion and anomaly detection project. Project development is nearing R&D completion. The anomaly detection model is discussed as a layered set of interface specifications for deriving near-real-time warnings based on analysis of audit events in a heterogeneous network environment. Various implementation features of the ISOA are also discussed; these include: monitoring structures, the statistical approach, and the user interface. The monitoring structures of the ISOA support a hierarchical analysis and flow of data from raw audit events through perceived security situations and the automated generation of warnings based on monitored sessions.

Introduction

This paper presents an updated theoretical and functional overview of the ISOA, which is nearing completion of the R&D cycle. A number of versions of the system are currently installed and used outside of the laboratory. Developed since 1985 by our Research and Development staff, the ISOA is a state-of-the-art system for automated intrusion and anomaly detection. The ISOA was designed to serve as a harness for both existing and emerging techniques and technologies. A number of innovative features in the current version of the ISOA prompt us to submit this paper.

Typically, security monitoring of user activities—along with detection of anomalous behavior—can be based on analysis of audit and transaction data generated by operating systems, data base management systems, etc. The huge volume of this data mandates automated analysis because manual examination of the audit trails is too slow and consequently too costly. If automated analysis tools are unavailable or deficient, audit trails are typically ignored until after a violation has been detected and, usually, some damage has been done. The ISOA can support both real-time and batch analysis to identify unusual and/or suspicious behavior. The ISOA addresses the following specific needs:

- Reducing the storage volume for audit data;
- Timeliness of automated audit analysis;
- Ease of use with minimal user involvement;
- A robust model for security monitoring;
- Simplified porting to fundamentally new environments;

- Ability to respond to an evolving variety of threats and situations.

The ISOA is capable of detecting insider threats, intruders, and suspicious transactions. Detection of suspicious transactions involves identifying specific transaction parameters that are either inherently suspicious, suspicious due to an observed level of activity, or suspicious in the context of other parameters.

A typical target environment can consist of a combination of audit-generating hosts, servers, and workstations. Individual workstations may or may not generate audit records for analysis. In addition to security relevant audit records, the monitored environment will most likely also generate a variety of additional records of activities. These will typically include system accounting information and, potentially, transaction logs or records. These additional sources of security relevant information often are necessary to achieve a broader perspective than can be derived from audit trail analysis.

Anomaly Detection Model (ADM)

The authors propose that in the field of intrusion/anomaly detection we are lacking a scalable methodology for collecting, analyzing, representing, and processing audit information. Although various intrusion detection systems exist today [1,2,3,4,5,6,7,8,9,10], our experience and/or knowledge of these leads us to believe that typically they were designed for specific environments. This tends to make adapting or porting existing systems to new target environments a time consuming effort, possibly entailing significant software changes.

Our survey of the literature indicates that existing intrusion and anomaly detection systems have been implemented without benefit of a sufficiently general underlying conceptual model. While we do not contest the importance of the seminal work performed by the implementors and researchers in the field [1,2,3,4,5,6,7,8,9,10]—indeed we are indebted—we have found that these systems include fundamental and common levels of data mapping and data analysis. We have organized these into a model for intrusion and anomaly detection, the Anomaly Detection Model (ADM). The ADM defines various stages and aspects of processing audit data; it defines a hierarchical analysis strategy; and it links its results back to the collection level. We present our model, the ADM, as an extensible framework for intrusion/anomaly detection. Figure 1 depicts the various levels of processing in the ADM, these are:

- **Data Collection:** Audit records (along with other security related information regarding monitored assets) are made available to the monitoring entity. Activities are limited to collecting, converting, and registering raw audit records.

- **Data Organization:** At this level the collected data are organized in such a way as to facilitate further processing. Event data are categorized and associated with the appropriate internal mechanisms. Basic statistical accumulations are computed.

- **Synthesis:** Various kinds of measures and intermediate results are derived from the outputs of the previous level. This can involve various statistical techniques, neural net filtering, genetic algorithms, etc. Many activities at this level may be done periodically rather than every time an event record is received, thus enabling the use of more complex processing techniques while maintaining adequate throughput to keep up with the rate of incoming data.

- **Assessment:** At this level, the outputs of any and all previous levels are assessed to determine their security implications, i.e. the *meaning* of lower-level information is determined. This task is appropriate for a rule-based or knowledge-based expert system.
- **Response:** Based on the results of lower processing levels, the system responds appropriately to the current situation. The chief types of responses are feedback, warning, reporting, and countermeasures. Feedback can control processing at any lower level of the ADM—for example: increase audit granularity or depth of analysis, or initiate automated proactive investigation.

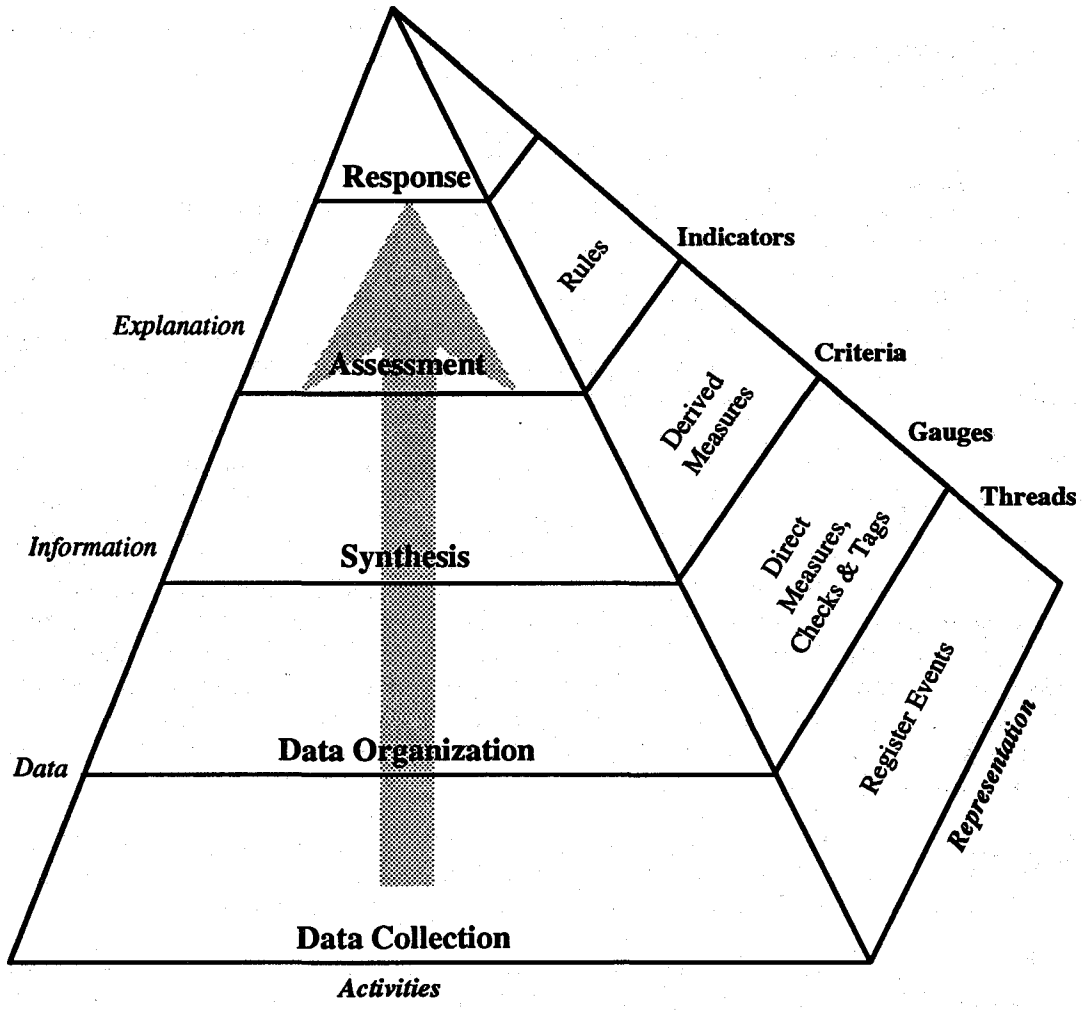


Figure 1 — The Anomaly Detection Model (ADM)

Incoming data are translated and analyzed in various ways as they flow upward through the ADM levels. Thus a large volume of raw data introduced at the bottom level is transformed into a manageable volume of meaningful information; the security implications are assessed, and appropriate responses are triggered.

The ADM provides a methodology for deriving meaning from the vast number of related and unrelated events which arrive over time. In an implementation, this entails maintaining

an abstract view of the current security relevant activities of each monitored entity (e.g. host, user, process).

Intelligent control of the audit analysis process by feedback mechanisms provides for efficient utilization of the available processing capacity. Analysis can be focused on monitored entities currently having higher security concern levels, thereby increasing the probability that a threat will be detected. Conversely, unnecessary processing can be avoided where it is unlikely to yield valuable results.

In summary, data must be collected and organized to facilitate further processing and interpretation. Analysis can be performed in a variety of dimensions. At the lowest level, it is necessary to recognize the occurrence of outright violations. At higher levels, one can perform statistical and rule-based analyses. Results from lower levels are made available to higher levels, resulting in an evaluation of the significance of the information—and an appropriate response.

Monitoring Structures

In the ISOA implementation, we use a series of linked structures which dynamically store information about monitored entities. These structures are identified on the right side of the ADM pyramid as:

- Threads— directly record events related to a particular monitored session;
- Gauges — defined as sets of numerical registers, with at least one set per currently monitored entity. Each currently monitored session has an associated set of gauges which store the current numerical values resulting from analysis. (Gauges are discussed further below);
- Criteria — boolean interpretations of gauges. We use the term *criterion* to refer to a distinct event or analysis sub-product, with each criterion having some expected, current, and trigger value;
- Indicators — a given criterion can be included in multiple *indicators* in the form of a simple or complex logical expression. Indicators include action lists which are invoked when indicators are triggered.

Numerical results of analysis are represented by a set of *gauges*, allocated to each entity being monitored. Each gauge represents some kind of quantitative information about a session component (an event type, a set of event types, or some statistical analysis of a session component). Gauges are continually updated by various ISOA processes, providing a representation of current and recent activity. Each gauge is owned by a particular process, which updates it. Any other process can use the value of the gauge in its internal computations. For example, one process can update a gauge used as a counter for file accesses, and another process can subsequently read the file-access count gauge, derive statistical measures from it, and post its results in other gauges. Because gauges are stored in shared memory, these exchanges are highly efficient.

The gauge representation allows information from different sources and with totally different meaning to be handled in a standardized manner for compatibility between the different processes. Any or all of the gauge definitions can be changed in order to tailor the ISOA to the environment being monitored.

Criteria are defined to reference gauges and define interpretations of the gauges for use by the expert system component of the ISOA. Any gauge may be referenced by multiple criteria, each defining a different interpretation of the significance of the current value of the gauge. A given criterion is *triggered* when the value of the gauge satisfies a condition specified in the *criterion definition*, such as exceeding a threshold value.

Likewise, one or more criteria are associated by a rule structure in an *indicator definition*, which is evaluated as a boolean combination of the current states of the component criteria. The state of a given indicator is a function of the state of its set of criteria. An indicator is *triggered* when the condition defined by its rule is true.

Some criteria and indicators may remain in the triggered state for a predefined period of time; others may be reset when trigger conditions are no longer true.

The complete set of criterion and indicator *definitions* constitutes a rule base which is used to interpret audited events as they occur. The *states* of the criteria and indicators for individual monitored entities serve as flags representing the current security status. At the lowest level we analyze raw audit records, while at successively higher levels we examine related events, sequences of events, sessions, and trends. The use of gauges, criteria, and indicators facilitates mapping analysis products from lower levels of the hierarchy to the highest levels.

Statistics

In order to reduce the computational load of statistical calculations, we have been investigating statistical techniques which minimize the amount of processing required for each and every audit record as it is received. We register the occurrence of the event when the audit record is received, and then periodically update other statistical measures based on the information recorded when the audit record was received. This approach allows the system to handle a greater volume of audit data, because the more complex statistical analysis tasks are spread out at wider time intervals. The frequency of statistical updates can be tuned to the needs of a particular installation. If the audit volume is not very great, a higher frequency can be used to enable quicker response to potential threats. If the volume is high, a lower frequency can be used to reduce processing overhead.

The most basic kind of statistical analysis involves counting the occurrences of various types of events in the audit stream and measuring rates of occurrence. All information indicating occurrence of discrete events is analyzed similarly. The same method can also be applied to some other kinds of information, if the value is monotonically increasing or decreasing (e.g. cpu usage). ISOA calculates short-term and long-term rates of occurrence with respect to user connect time. Other reference scales might also be used, such as host up-time. The rate measures may then be compared (at the criterion level) with stipulated thresholds or with historical frequency distributions.

Time values used for statistical calculations are obtained from the audit stream rather than from the ISOA system clock. Because of this, it is possible to replay a stored audit trail and run the same statistics program that is used during real-time monitoring, even though the replay can proceed as fast as the processing capacity of the ISOA will allow.

Statistical values are recalculated at a frequency low enough to avoid excessive processing overhead, yet high enough that significant anomalies will be detected soon after they occur. This frequency is tuned to the needs of a specific installation.

Values of statistical measures for each user are periodically written to disk files (minimally, at the end of each session). At the beginning of a session, the appropriate values are loaded from the disk file to initialize the data structures used in the statistical computations. Thus, it is possible to perform real-time session analysis, based in part on previously monitored sessions, derived measures, and so forth.

Exponentially Weighted Rate Measures

Rate measures are calculated by an exponential formula which gives more weight to recent events than those in the more distant past. Each rate measure is based on a half-life; the reciprocal of the half-life is the data decay rate. The half-life is the length of time in which the rate measure will decay to half its current value if the associated event does not occur in that time. This is similar to a technique used by IDES [4,9,11]; however there are a number of differences. In the ISOA, we are using multiple rate measures for the same event type, each having a different half-life. If the event type in question occurs at a constant rate, all rate measures for that event type, regardless of half-life, will eventually stabilize at the same value, indicating the rate of occurrence of the event. Rate measures with short half-lives will respond quickly to short-term changes in the rate of occurrence of the associated event; those with long half-lives will give an indication of the average rate over a longer period of time. The system is normally configured to use the same set of half-life values for every audit event type. This enables direct comparison between rates for different event types and for different users/hosts. The rate measures are expressed in units that are meaningful to a human analyst—normally number per hour. This is helpful for interactive investigation of suspicious situations. Also, since the rate measures are on the same scale, regardless of the half-life, they are directly comparable. Thus, a simple graphic display of the set of rate measures for a particular event type can give the ISO an instant picture of the recent history for that event type (figure 2).

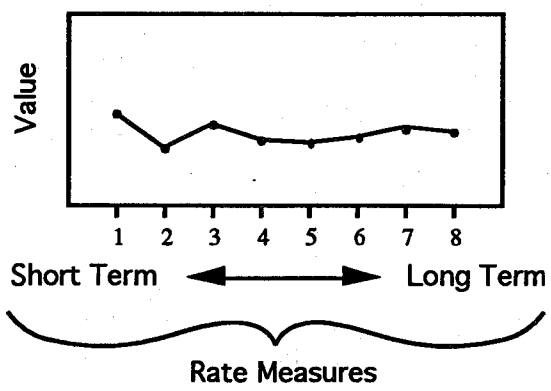


Figure 2 — Rate Measure Graph

Event Sets

Since we use multiple rate measures, the short-term ones are a very specific indication of *current* activity, say, within the last minute. By comparison, in a system based on a single rate measure for a given event type, the half-life must be adjusted to take into account a much greater amount of time; otherwise the erratic behavior of the measure would make it useless. In the multiple-measure scheme, we actually exploit this erratic behavior by comparison to less erratic measures (with longer half-lives).

Use of two rate measures with very short half-lives, such as one minute and two minutes, enables a powerful fuzzy-logic correlation between different events. If we look at a short-term rate measure compared to the next longer term measure, a ratio greater than one indicates a recent increase in this measure. The set of event types showing an increase at any given time represents a particular pattern of user activity. These patterns can be represented as bitmaps, which makes comparisons very efficient. We would expect that only a small fraction of all the possible patterns would actually be represented. We can keep track of which patterns have actually been observed for each user. Also, we can distinguish between patterns which have simply been observed in the past and those which are administratively recognized as normal. In other words, a pattern should not automatically be considered normal just because it has been previously observed. Also, a completely new pattern need not generate a warning immediately; instead, it can trigger selective automated analysis in order to determine whether a warning is warranted.

Here is a more formal description of the set computation: Let $E_1, E_2,...E_m$ be event types. For each event type, let $R_1, R_2,...R_n$ be rate measures, from short-term to long-term. Let $Q_{i,j}$ be the quotient R_i / R_j. Let S_t be the set of event types E, such that $Q_{1,2} > 1$ at time *t*. Now let S' be the set of all sets S_t for a given time range. S' is the *pattern* described above.

If all possible sets S_t were actually observed, S' would be the power set of the set of all event types. This set would be extremely large for large m (cardinality 2^m). We expect the actual S' to be comparatively small. If it does grow too large, it can be pruned by eliminating event sets which have been very rarely observed. If pruning is implemented, we expect to provide for manual override to prevent pruning of specified event sets.

Trends

Using multiple rate measures with different time scales provides a constantly updated indication of the trend of a measure. For instance, if the short-term measures have larger values than the long-term ones, there is an upward trend. Normally we would expect substantial fluctuation in short-term measures, with the magnitude of fluctuation decreasing with increasing half-life. Suppose we have rate measures $R_1, R_2,...R_8$ with half-lives of 1, 4, and 15 minutes, and 1, 4, 16, 64, and 256 hours (roughly increasing by multiples of four). R_8 has the longest half-life, so it represents the average rate over the longest time interval for which we have information. Normally, R_7 might be somewhat higher than R_8, but then R_6 might be lower, R_5 still lower, R_4 higher, etc. However, if we found R_7 higher than R_8, R_6 still higher, R_5 higher than that, etc., we could tell that there has been a steady upward trend.

Of course an exception to this interpretation must be recognized in the case where we start the system with all rate measures having zero values, for instance because we do not yet have a basis for an expected average value to use at startup. In this case, we have to keep track of how long we have actually been computing the rate measures, and perhaps extrapolate to produce synthetic values for long-term measures. This could be done efficiently by means of a formula or table yielding a percentage adjustment figure based on the half-life and the actual time base of observed events.

User Interface

The user interface supports the display of audit-record-derived information in both textual and graphical representations. The chief elements are:

- Overall system control: system control, system mode, system feedback (located at the top of the display);
- Graphic network representation: security status display (located at mid-left of the display);
- Audit traffic window: raw audit record display with various text search capabilities (located at the mid-right of the display);
- Intervention capabilities: user id prompt, host prompt, etc (located horizontally under the main system control area of the display);
- Analysis feedback and control: expert system feedback and user-directed control (located at bottom left of the display);
- Statistics, plot and data visualization (located at bottom right of the display);

Within each of these windows, various lower-level functions are accessed via pop-up windows and information displays. This approach facilitates user/system interaction, reduces complexity, and allows the integration of further capabilities. The lower-level facilities available to the ISO include the profile editor and rule editor. The profile editor permits the ISO to specify expected behavior parameters at the granularity of a single host or user. The rule editor allows dynamic modification and definition of rules.

At any time during audit analysis and security monitoring, the current perceived security status of all hosts and sessions is represented by a graphical display of the monitored network. This window, the GRAPH display, is color coded to provide a clear indication of the highest level of concern for all sessions on a given host. A black GRAPH display indicates no activity, green indicates acceptable activity, yellow indicates a low-level warning, and red indicates cause for serious concern. In addition, the ISO can open a pop-up display window for each monitored host. This dynamically updated window gives a detailed synopsis for each user session for that host.

Analysis of suspected security threats requires easy access to various kinds of data, such as file statistics, command usage statistics, and profile threshold values. A graphical display capability is virtually mandated by the importance of recognizing patterns and relationships in the data. To meet this need, a data plotting utility has been developed for the ISOA. While the user interface and plotting is done by a separate process (PLOT), the data collection for plots may be performed by other processes, which then pass the data to PLOT via shared memory. Mouse-sensitive areas in the plot windows allow the user to selectively display additional information about parts of a plot. Textual annotation in plot displays is thus kept to a minimum, making the graphic display less cluttered and therefore easier to read. Dual plot windows allow comparison of related plots, or simultaneous display of two different, possibly unrelated, plots. Each plot window has its own control panel to select the data to be displayed.

During a typical monitoring session the ISO will periodically check the status of the ISOA's analysis and warning capabilities. The ISO does not need to maintain visual contact with the ISOA since a complete record of warnings and generated analysis information are

recorded in a scrollable window for review by the ISO. The end result of anomaly resolution is presented to the ISO in the form of a graphical alert with system generated advice and an explanation as to why the expert system has set the current security concern level. The graphical interface includes numerous other windows for monitoring audit traffic, directing control of the ISOA system, and effecting direct control of monitored user sessions and hosts. When monitoring indicates anomalous activity on a given host, the ISO can obtain more in-depth information by selecting a graphical representation of that host. As described previously, graphical representations of monitored hosts are color coded to depict their current security status.

Conclusion

The ISOA has been under development for the past five years. At this time we are completing our R&D development efforts. Current activities include restructuring the GUI layer to ease porting to different window systems. The ISOA currently runs on Sun workstations under Sunview and IBM RS6000 platforms under Motif. The same software can be moved to other UNIX platforms with minimal changes. The system consists of some 50,000 lines of 'C', and includes no third party software. Currently, the ISOA is running in three environments, two outside our R&D development facility.

At this time we are tracking various trends in the intrusion and anomaly detection community, including initial prototype efforts in applying neural network technology to this problem area. We believe that the fundamental difficulty in the intrusion/anomaly detection area is in the generation of meaningful audit information by operating systems and applications environments. We also recognize the utility of incorporating non-security-domain information such as network traffic analysis and accounting and systems information generated by most operating systems. Requirements for security monitoring will most certainly vary among different environments. We look forward to the availability of a variety of tools which are capable of cooperating and can be combined to monitor and analyze behavior in complex network environments.

References

[1] Anderson, J.P. 1980. "Computer Security Threat Monitoring and Surveillance". James P Anderson Co., Fort Washington, PA, April 1980.

[2] Denning, D., "An Intrusion-Detection Model", *Proceedings of the 1986 IEEE Symposium on Privacy and Security*, April 1985.

[3] Bishop, M., "A Model of Security Monitoring," *Proceedings of the 5th Annual IEEE Computer Security Applications Conference*, December 1989.

[4] Lunt, T.F., "Automated Audit Trail Analysis and Intrusion Detection: A Survey", *Proceedings of the 11th National Computer Security Conference*, October 1988.

[5] Bauer, D.S and Koblentz, M.E., "NIDX - A Real-Time Intrusion Detection Expert System", Proceedings of the Summer 1988 USENIX Conference, June 1988.

[6] Halme, L. R. and Kahn, B. L. 1988. "Building a Security Monitor with Adaptive User Work Profiles." *Proceedings of the 11th National Computer Security Conference,* October 1988.

[7] Sebring, M. M., Shellhouse, E., Hanna, M. E., and Whitehurst, R. A. 1988. "Expert Systems in Intrusion Detection: A Case Study." *Proceedings of the 11th National Computer Security Conference,* October 1988.

[8] Vaccaro, H.S., and Liepins, G.E., 1989. "Detection of Anomalous Computer Sessions Activity", *Proceedings of the 1989 IEEE Computer Society Symposium on Security and Privacy.*

[9] Lunt, T. F., Jagannathan, R., Lee, R., Listgarten, S., Edwards, D. L., Neumann, P. G., Javitz, H. S., and Valdes, A. 1988. *IDES: The Enhanced Prototype, A Real-Time Intrusion-Detection Expert System.* SRI-CSL-88-12. Menlo Park, CA: SRI International, Computer Science Laboratory.

[10] Clyde, A.R., "Insider Threat Identification Systems", *Proceedings of the 10th National Computer Security Conference*, September 1987.

[11] Lunt, T.F., Tamaru, A., Gilham, F., Jagannathan, R., Jalali, C., Neumann, P.G., Javitz, H.S., Valdes, A., Garvey, T.D., 1992. "A Real-Time Intrusion Detection Expert System (IDES)—Final Technical Report", SRI International, Menlo Park, CA, February, 1992.

[12] Dias, G.V., et. al., "DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and An Early Prototype" *Proceedings of the 14th National Computer Security Conference,* October 1991.

ISSUES IN THE SPECIFICATION OF SECURE COMPOSITE SYSTEMS

Judith Hemenway
Grumman Data Systems
4015 Hancock Street
San Diego, CA. 92110
Hemenway@dockmaster.ncsc.mil

Dan Gambel
Grumman Data Systems
2411 Dulles Corner Park
Herndon, VA. 22071
Gambel@dockmaster.ncsc.mil

1.0 Introduction

This paper presents a number of issues in the specification of secure composite systems which have emerged from our efforts to design and build secure distributed systems for "real-world" applications. These issues are identified and discussed here in order to heighten awareness of the difficulties and complexities entailed in specifying and verifying secure composite systems, and to stimulate further work in the area of theoretical models for system composition. In this introductory section we discuss briefly why system composition is a significant issue, what the current state-of-the-practice is with respect to composite systems, and why we consider formal specification to be particularly relevant to system composition. In section 2 we present the specification issues that we have identified, provide examples to illustrate each issue, and determine whether any of the specification approaches that we have surveyed can be used to address the issue. Section 3 provides a brief summary and future plans.

1.1 Trends in Computing Configurations

The trend away from centralized computing systems to distributed systems is firmly established in both the commercial and Government domains. In addition, the need to minimize development and maintenance costs has led to increased efforts to utilize existing commercial off-the-shelf (COTS) equipment wherever possible. As a result of these two trends, it is apparent that many future computer systems will utilize networks to integrate COTS components from multiple vendors. Secure applications will need to incorporate COTS component level TCB technology. The use of multilevel secure (MLS) workstations to provide graphical user interfaces will be widespread, as will be the use of distributed MLS servers which support a variety of distributed processing.

Secure commercial and military system components are increasingly available. Worked examples of multilevel operating systems, database management systems, and local and wide area networks can be found in the open literature. The NCSC's Evaluated Products List contains approximately 10 entries that have received B- or A-level ratings, and approximately 10 additional products at the B-level or above are currently undergoing design analysis or formal evaluation. Today, the critical missing technology is the ability to create a composite system, using as components a heterogeneous collection of existing products, only some of which may be secure themselves.

1.2 Current State-of-the-Practice

Although, as noted above, there are an increasing number of secure products available at the B-level or higher, there are as yet very few worked examples of composite systems which incorporate such products. For the purposes of this paper, we have chosen two example systems for which the authors participated in design and implementation. These systems serve as examples to illustrate points made in the remainder of the paper. The two systems chosen for discussion are Network Reference Monitor (NRM), a wide-area network targeted at the A1-level [Fell87], and Headquarters System Replacement Program (HSRP), a C2-targeted data processing system integrated from COTS components, and designed to be migratable to B-level assurance [Gamb90].

The NRM system provides MLS end-to-end encrypted communications across a packet-switched

network. NRM comprises three types of components: a centralized access control center, a centralized key generation and distribution facility, and a set of network front-ends, one for each host connected to the network. In NRM's view, the subjects are the subscriber hosts, and the objects are the crypto-connections between pairs of hosts. Both multi-level and single-level hosts are permitted, but all connections are single-level. Each NRM component consists of a general-purpose MLS operating system supporting a collection of special-purpose processes, some of which are trusted. The trusted processes may be either single-level or multi-level with respect to the operating system. Thus, each component has its component-local Trusted Computing Base (TCB) consisting of the operating system and some of the trusted processes, and the system or network TCB (NTCB) consisting of all of these local TCBs plus the remainder of the trusted processes. In this context, "trusted" process means one of three things:

1. With respect to the local OS, the process must handle multi-level data.
2. With respect to the local OS, the process is single level but performs some security-relevant function for the local TCB (auditing, for example).
3. With respect to the local OS, the process is single-level but performs some security-relevant function for the NTCB.

It should be noted here that much of the security-relevant data and decision-making for the system-level security policy is contained in the special-purpose processes of the components, rather than in their operating systems. It should also be noted that the NRM components were all developed from scratch, concurrently, by one vendor, so it did not suffer from some of the multi-vendor incompatibilities to which composite systems are prone. However, although it was developed as a system (the "single trusted system view" of the Trusted Network Interpretation [TNI87]), it is designed to fit as a component in larger systems (the "interconnected accredited AIS view"), where incompatibilities are quite likely to occur.

The HSRP system is a true composite trustworthy system, in that it is composed of a collection of three types of components developed by different vendors: a set of single-level mainframes which provide discretionary access control (DAC) enforcement, a MLS terminal multiplexor, and a centralized authorization center in a local area network (LAN) environment. In this system, with respect to the mandatory access control (MAC) policy, the subjects are individuals sitting at the terminals, and the objects are the single-level mainframes. Both the terminal multiplexor and the authorization center components are composed of a general-purpose MLS operating system and a collection of special-purpose processes, some of which are trusted. Again, there is a distinction here between component-local TCBs, and the overall NTCB.

The composition of the HSRP system can be viewed as occurring in three stages. In stage 1, a collection of software components (some COTS, some newly developed) is integrated to form each type of physical component (mainframe, terminal multiplexor, and central authenticator). In stage 2, the physical components are combined (conceptually) to form the Mandatory (M), Discretionary (D), Identification (I) and Audit (A) components of the TNI. Here all the mainframe components together form a TNI DIA component, the terminal multiplexors together form an MA component, and the central authenticator provides an IA component. In stage 3, the DIA, MA, and IA components are composed to form the overall HSRP system which provides all four functions. Alternatively, stage 2 could be viewed as an individual function composition, such that portions of the mainframes are composed to form the D function, portions of the terminal multiplexors to form the M function, portions of the central authenticator and mainframes to form the I function, and portions of all three (mainframes, terminal multiplexors, and central authenticator) to form the A function. In stage 3, then, the four separate functions are combined to form a complete system.

1.3 Role of Specification

The role of formal specification and verification in the development of secure systems has, over the past two decades, gradually progressed from a research topic, to application on small monolithic

systems, to occasional use as a real development tool on large-scale real-world systems. Wide-spread acceptance of formal methods has not yet been achieved, in part due to the expense of using such techniques, and in part due to misperceptions of what such techniques can actually do. The assumption of the TCSEC is that formal methods provide additional assurance above and beyond the assurance that can be achieved by more traditional software engineering methods, and that such methods are highly desirable where high levels of assurance are required. Thus, a formal security policy model is required at and beyond B2, while at the A1 level a formal top-level specification is also required.
Our experiences with formal methods on a number of system development and integration efforts support the value of formal methods. We have found that their use encourages more rigorous thinking very early in the design process, resulting in identification and resolution of potential design problems. We have also found formal methods useful during implementation as an additional means for identifying both security-relevant and non-security-relevant implementation errors, during both code walkthroughs and testing.

Given the current trend toward networks of COTS components, it is clear that future secure systems will be increasingly complex due at least in part to their distributed nature, thus introducing potentially increased vulnerability to both accidental and malicious threats. It is within this context that we think formal methods can and should be applied as one means of controlling the increased complexity, and minimizing potential vulnerabilities. Considerable research is currently being conducted in the area of composition of systems/specifications. As part of this study, we have surveyed this research and identified a number of approaches that we consider applicable to the problems that we face as integrators of secure systems. In the next section of this paper, we explore what problems there may be in trying to apply these approaches to real-world system composition efforts.

2.0 The Issues

The approaches we have surveyed provide a wealth of concepts and viewpoints relating to the specification of distributed systems. The approaches differ with respect to how system components are inter-related and composed, what properties may be expressed and proven for the resulting system, and whether the overall viewpoint of the specification is external (i.e., descriptive of the interface) or internal (i.e., descriptive of internal states).

In the discussion below, we describe a set of issues which have been identified in the course of our experiences in specifying secure distributed systems. For each issue, we provide a description of the issue, one or more illustrative examples, and a discussion of how the issue might be addressed using particular specification approaches.

2.1 Component Roles

The functions performed by a TCB include not only access control decisions, but also maintenance of the data related to access control (e.g., user IDs and clearances, access control lists, object labels, etc.), user identification and authentication, and auditing of security-relevant events. In a distributed TCB, such functions and data may be replicated across the components of the system, or partitioned among the components.

Replication across components: In the NRM system, the ID and security range of each subscriber host (subject) are maintained both on the access controller component, and on the front-end component which attaches the host to the network. In HSRP, establishing the identity of a user for DAC purposes is performed by each mainframe that a user logs onto.

Partitioning among components: In NRM, the current access set consists of a list of currently active crypto-connections. Since each front-end component maintains a list for only those crypto-connections where its host is one of the end-points, this means that the current access set is partitioned among the front-end components, as is the access control decision-making for the use of those connections. In

this arrangement, the access controller component performs the function of granting access (i.e., establishing the connection), while the front-end component controls the use of the connection. A different type of partitioning is illustrated by the HSRP system. Here, the central authenticator performs identification and authentication functions, while the terminal multiplexor performs the MAC function, and the mainframes perform DAC. This is the type of partitioning of function that is addressed in the TNI, with its M, D, I, and A components. As another example, in the HSRP system, DAC is partitioned among the mainframes, with each mainframe having its own set of named objects to which it controls access, based on the user's identity.

In a somewhat more complicated example in HSRP, a terminal profile table is partitioned among the terminal multiplexors, with each multiplexor having profiles for only those terminals attached to it, but the central authenticator maintains a complete table of profiles for all terminals. Thus, here we have a combination of partitioning and (partial) replication.

Specification Implications: Replication of data across system components requires the implementation of some mechanism(s) to ensure consistency of that information. Replication of function may also require some means of ensuring consistency: for example, it may not be consistent with the overall policy for one component to permit a user to write above his/her clearance level, while another component does not permit such "write-ups". Partitioning of data typically does not require any consistency mechanisms, but it would be desirable in the specification to have some means of describing how the whole is partitioned (or how the partitions are composed). Partitioning of functions (particularly in the sense of the TNI) typically requires the establishment of protocols by means of which the various components can exchange information and coordinate their actions.

It would appear that the specification of component roles in distributed systems involves two requirements: the ability to specify protocols for maintaining consistency and coordination among components, and the ability to specify partitioned data. Protocol specification has been addressed in work such as Hailpern and Owicki's [Hail80]. As for the composition of partitioned data, the concept of state expansion discussed by Abadi and Lamport is a possible solution [Abad90].

2.2 Inter-Component Dependencies

Shockley [Shoc90] provides a definition of domain (component) "dependence", and observes that two components may be either independent, mutually dependent, or unilaterally dependent with respect to some set of correctness criteria: Given two domains, dA and dB, specifications of their interfaces, sA and sB, and demonstrations of implementation correctness, vA and vB, Shockley asserts that "Domain dA 'depends (for its correctness)' on domain dB if and only if the arguments within vA assume (in whole or part) the correctness of the implementation of dB with respect to sB as a premise." This definition has also been adopted in the TDI as the definition of TCB subset dependence (i.e., TCB layering). Note that where more than two components are involved, unilateral dependence may be either circular or strictly hierarchical (partial ordering).

Independent components: In NRM, the front-end components are all independent of each other with respect to enforcement of the security policy. That is, no front-end component depends on any property provided/enforced by any other front-end component in order to enforce the security properties correctly. A similar situation occurs in HSRP, where the mainframes are independent of each other with respect to enforcement of DAC, and the terminal multiplexors are independent of each other with respect to enforcement of MAC.

Mutually dependent components: For the NRM system, the access controller and the key distribution components are mutually dependent: the access controller depends on the key distribution component to provide keys for only those connections that the access controller has approved, while the key distribution component assumes that the connections it has been directed to provide keys for are valid connections (with respect to the security policy). Note that, for each established connection, the front-end components at each end of the connection are mutually dependent with respect to successfully

providing communications between their attached hosts (a service assurance policy) while, as noted above, they are independent with respect to MAC enforcement. In the HSRP system, mutual dependence exists between each terminal multiplexor and the central authenticator: the terminal multiplexor relies on the authenticator to perform identification and authentication of users and to calculate the range intersection for a user-terminal pairing, while the authenticator relies on the terminal multiplexor to determine the user-terminal pairing and to permit access only to authenticated users.

Unilaterally dependent components: Each front-end component in the NRM system depends on the access controller to provide the ID and security range of the host attached to the front-end component. In HSRP, each mainframe depends on the terminal multiplexor to supply the user's login ID. In both of these examples, the dependence is strictly hierarchical. Another example, in both NRM and HSRP, is the dependence of the trusted processes on the services provided and properties enforced by the underlying operating system (a classic example of TCB layering). We elaborate further on this concept in section 2.6 on secondary/supporting policies.

Specification Implications: Communications protocols play an important role in distributed systems, not only for user communication, but also for communications among the distributed portions of the NTCB. The typical protocol "stack" represents a strictly hierarchical component structure, where each protocol layer is a component. In this case, dependence is similar to Lam and Shankar's (see [Lam91a] and [Lam91b]) concept of linear hierarchies of modules in which a module "uses" the interface of a lower module, and "offers" an interface to a higher module. (Note also, however, that the peer entities in a protocol stack may be mutually dependent.) In addition to protocol specification, an approach such as Lam and Shankar's may be very suitable for situations such as those addressed in the TDI (where the concern is for incremental evaluation or "evaluation by parts"), and for extensible architectures such as described by Schaefer and Schell [Scha84]. Hoare's CSP [Hoar85] permits both sequential composition (unilateral dependence) and parallel composition (mutual dependence and independence), as does the Abadi-Lamport Composition Rule [Abad90].

2.3 Granularity of Elements

It is frequently the case within distributed systems that the granularity of both subjects and objects is widely variable across components, and, further, that the subjects or objects of one component must be maintained or controlled in some specific relationship (including a relationship of labels) to the subjects or objects of other components.

Subject granularity: In the NRM system, the subjects are the hosts attached to the front-end components. However, it is quite likely that when NRM is installed as the network component of some system consisting of a collection of MLS hosts, the granularity of subjects on each host will be individual processes. Thus, what is a single subject from the point of view of the NRM security model is actually a set of subjects from the point of view of the model for the enclosing system. A somewhat different situation occurs in HSRP, where the subjects are processes on the terminal multiplexors and on the mainframes. In this instance, it may be desirable to have a way of expressing the fact that subjects A and B on the terminal multiplexor, and subjects X, Y and Z on the mainframe are all related in that they represent the same user, operating at the same security level. We call such a relationship a "federated" or "session" subject.

Object granularity: In the NRM model, the objects are the crypto-connections between pairs of hosts, and sending a message via a connection is considered "modifying" (in Bell La Padula terms) while receiving a message is viewed as "observing". An alternative model could view crypto-connections as containers, with messages being the elementary objects contained in the connections. Such an approach would be similar to the compound objects defined in the MMS model [Land84]. In the HSRP system, the objects controlled by the terminal multiplexor are the mainframes, whereas the objects controlled by the mainframes are the more traditional objects such as files, messages, segments. From one view, this is the TCSEC difference between MAC "objects", and DAC "named objects", while from another point of view, this is another instance of a single (container) object consisting of a set of finer-grained

objects.

Paradigm shift: A more subtle distinction with respect to granularity is the issue that Bell has termed "paradigm shift". To illustrate, we return to the NRM notion of a crypto-connection as an object, with "observing" the object being defined as receiving a message via the connection, and "modifying" the object being defined as sending a message, as described above. A crypto-connection here is really an abstraction, rather than a physical object, and is represented by a connection record, which is a collection of information about the connection. This connection record is stored in a table in the memory of the front-end component, and access to this table is controlled by the operating system of the front-end component. Thus, with respect to the system model, a connection is an object, access (i.e., sending and receiving messages) to which is controlled by the trusted software which resides in the front-end component, running on top of the operating system. However, with respect to the local-model for the front-end component, the table containing the connection record is the object, access to which (i.e., reading and writing the table) is controlled by the operating system. There is clearly a relationship between the two objects, but it is not a simple set/subset relationship; rather, it is a form of unilateral dependence resulting from the layering of a NTCB process on top of a trusted operating system (local TCB). We will return to this issue in section 2.6 when we discuss secondary/supporting policies.

Specification Implications: Many of the specification approaches we have investigated are based on an external view of systems and components, in which only the interface events and their associated "ports" are visible. Of the remaining approaches, which are typically state-machine descriptions, the MMS model [Land84] provides a form of object granularity by means of the concept of containers, and McLean's approach ([McLe88] and [McLe90]) provides for one form of subsetting of subjects in his definition of a subsystem. He also provides, in his framework for N-person rules, a definition of compound subjects, where each subject is a subset of subjects. Although these approaches do not accommodate all of the examples described here, they do provide a starting point for further work in the representation of variable granularity of subjects and objects.

2.4 Security Labels

In composite systems, the amount of information encoded in security labels, and the particular form of the internal representation of those labels, will quite likely vary from component to component. The TCSEC requires a minimum of 16 hierarchical levels, and 64 non-hierarchical categories, although some systems have implemented considerably more than that; for example, the new AT&T System V Rel. 4.1/ES supports 246 hierarchical levels and 992 categories. Even in situations where the number of levels and categories is the same for two components, the meanings assigned to the various levels and categories may differ. Reconciling such label inconsistencies during the integration of a composite system composed of pre-existing components is a critical and sometimes very difficult task, which typically requires the creation of a label translation/mapping function.

Another labeling issue arises in systems where the granularity of subjects or objects varies, as described in the previous section. It may be desirable or even necessary to enforce a particular relationship among the labels associated with a set of subjects or objects.

Label consistency: In the NRM system, all of the components were developed together, and shared a common label syntax and semantics. It was, however, necessary to provide a conversion function between the format used internally by the components and the format used in the IP Security Option (IPSO) defined at the interface with the attached hosts. Any hosts that attach to a NRM controlled network will likewise have to reconcile their own label definitions with the IPSO. Thus, the IPSO serves as an "intermediate" label form. Although HSRP was developed as a composite system, the architecture is such that label consistency was not an issue. This is due to the fact that all MAC is performed by the terminal multiplexors (with support from the authenticator), all of which are implemented on the same platform with the same OS, and each mainframe is a single-level environment, without MAC labels. It was, however, necessary to devise a protocol for distributing the

semantic interpretation of the MAC labels across the terminal multiplexors and central authenticator.

Label granularity: In the NRM system, each crypto-connection is established at a single security level, and every message that is sent or received via a connection must be at the level of the connection. Thus, connections and messages are both labeled, and a strict equality relationship between the labels is enforced. Since the "messages" in this system are IP datagrams, an additional labeling issue arose due to the fact that IP datagrams can be fragmented at the next lower protocol level. In this instance, it was decided to disallow fragmentation for multi-level hosts, rather than devising a labeling convention for the fragments. However, such a solution would certainly be possible, and again the relationship between datagram and fragment labels would be equality. Another illustration is in the Compartmented Mode Workstation (CMW), where, for each object, a non-changeable "protection label" is maintained that must always dominate a floating "information label". This is similar to the MMS notion of container labels (the protection label) and object labels, although the CMW information label would be equivalent to the least upper bound of the MMS object labels, rather than to the object labels themselves.

Specification Implications: Of the specification approaches that we have reviewed, none addresses label consistency. Only the MMS model explicitly addresses label granularity, and this is done only for objects [Land84].

2.5 Security Policies

In a composite system, it is quite likely that two or more components will each have a stated security policy that controls access of subjects to objects. In such systems, both issues of policy conflict, and issues of policy composibility must be addressed.

Policy conflict: Policy conflicts arise between components if one component enforces a property which negates or weakens the policy of another component. One example would be a system in which component A enforces the Bell-La Padula *-property, which prohibits write-downs but not write-ups, whereas component B enforces a policy which prohibits both write-downs and write-ups. Depending on the particular system, this may be viewed either as a legitimate difference, or a serious policy conflict. Another example would be a system in which one component permits owner-users to modify the access permission matrix (the ACLs), but another component allows only the Security Officer to do so.

A concise specification of each component's security policy permits straightforward identification of conflicts such as these. However, analysis and resolution of the conflicts is not a technical issue, but rather a policy issue, which must be addressed by the DAA(s) for the system. In some instances, such policy conflicts may not be security weaknesses but rather a legitimate dual policy situation, while in other instances, the conflicts may indeed be weaknesses, and some modifications to the policy and/or its underlying mechanisms may be necessary as part of the system integration effort. In either case, once such analyses have been performed, one must then address the issue of policy composibility.

Policy composibility: Even in situations where the policies of individual components do not conflict, it may or may not be possible to compose the components into a single system with a system-level security policy. In general, three different types of composition may be needed. The first type is replicated policies: the same security policy is enforced in two or more components, which will be composed to form a system. In HSRP, the DAC policy is replicated across the mainframe components, and the MAC policy is replicated across the terminal multiplexor components. The second type is sibling policies: similar security policies are enforced in two or more components which will be composed to form a system. An example of this type would be a network consisting of a component running GEMSOS, a component running trusted Xenix, and a component running AT&T System V. The third type is a single distributed policy. NRM is an example of this type, with enforcement of its security policy distributed across the three types of components in the system.

Specification Implications: McCullough [McCu88] discusses composibility for systems in which each component enforces the same security policy (the replicated policy type of system). Even here, composibility is not guaranteed: non-interference is not composible for non-deterministic systems, but restrictiveness is. McLean ([McLe88] and [McLe90]) provides a framework for security policy models in which each model is distinguished by permissiveness for changing security labels (tranquility "violations"), which defines a form of sibling policies. It may be possible to extend this concept of frameworks to encompass other dimensions on which policies may vary. Both the Lam-Shankar approach ([Lam91a] and [Lam91b]) and the Abadi-Lamport approach [Abad90] permit the composition of components with arbitrary policies, which would be of benefit particularly for the third type of system (single distributed policy).

2.6 Secondary/Supporting Policies

Within any complex, modularized system (distributed or not), it is frequently the case that those components which enforce the security policy depend on other portions of the TCB to supply secondary or supporting policies. By this we mean not only such functions as auditing, but a variety of functions and properties which become security-relevant by virtue of the fact that correct enforcement of the security policy depends on the function or property. This situation is particularly in evidence in distributed trusted systems. The distribution of portions of the system TCB among two or more components of the system results in a need for communication among the components. Such communication usually requires mechanisms to establish a TCB-to-TCB trusted path, and protocols which provide consistency of distributed security-relevant data, and concurrency control of security-relevant actions. Further, in those situations in which portions of the TCB are organized hierarchically (i.e., as a layered TCB), the policies enforced by higher levels of the hierarchy frequently depend on the "correct functioning" of lower levels.

Trusted communications: In the NRM system, the fact that both trusted and untrusted components use the same communication medium results in the need for a mechanism that the TCB partitions can use to authenticate themselves to each other, and to protect from disclosure or modification the security-relevant data which they must exchange with each other. The use of pair-wise keying provides both authentication and data protection, which means that the TCB depends on the correct functioning of the keying protocol. In effect, encryption is the mechanism whereby TCB-to-TCB messages are rendered tamperproof while they are beyond the protection of hardware domains. The situation is somewhat different in HSRP, where the terminal concentrators are connected to the authentication center via an Ethernet that is for their exclusive use. Thus, no untrusted components have access to the communications medium, and so the need for protected communications is obviated. However, an identification and authentication protocol among the TCB components (terminal multiplexors and central authenticator) was implemented, primarily to protect against errors which could result in reaching a non-secure state.

Concurrency control: As was described in section 2.1 above, certain security-relevant data in the NRM system are replicated across two or more components. In situations such as this, it is necessary to use a protocol (such as a two-phase commit) to control updates to the data in order to maintain data consistency. A second example of concurrency control is the revocation of an established connection, where the actions of two components (the end-points of the connection) must be coordinated in order to complete the revocation. In the HSRP system, communication among the components assures that no user can view simultaneously two or more sessions that are operating at different security levels.

Hierarchical dependence: In section 2.3 above, we described the notion of "paradigm shift" for which the example given in NRM is that of a system level connection vs. a component level connection record. To reiterate: with respect to the system model, a connection is an object, access to which is controlled by trusted software which runs on top of a trusted operating system, whereas with respect to the component model, the table containing the connection records is the object, access to which is controlled by the trusted operating system. Thus, the higher level trusted software depends on the correct functioning of the lower level software in order to enforce its security policy. A similar

situation occurs in HSRP, where the authentication of users, and the MAC access decisions are performed by trusted software residing on top of a trusted operating system. A third example is the Separation VMM discussed by Kelem and Feiertag [Kele91]. Note that in all three examples, part of what the operating system does is to extend "tamperproofness" protection to include the trusted processes.

Specification Implications: Both trusted communications and concurrency control rely heavily on the use of protocols, for which approaches such as Hailpern-Owicki [Hail80] or Abadi-Lamport [Abad90] are quite useful. The issues of hierarchical dependence are more directly addressed by Lam and Shankar's approach ([Lam91a] and [Lam91b]).

2.7 Granularity of Execution

In a distributed system, actions which are traditionally modeled as atomic (uninterruptable) state transitions may actually require a sequence of such transitions, particularly where communication across a network is concerned. This constitutes an apparent shift in the "granularity of execution" when comparing the system viewpoint with the component viewpoint.

One example of this issue in NRM is the action of granting access to an object. Whereas in the traditional single-processor system, the granting of access is typically accomplished by means of a single system call (e.g., open file), the equivalent grant in the NRM system is accomplished by means of a protocol involving the interaction of all three types of components. Another example is updating the security-relevant data which resides in the access control component. Since the access control component may be replicated (e.g., for survivability), this action involves execution of a protocol among the multiple access control components, rather than a simple "update database" system call. Granting access (for MAC) in HSRP is similar to the situation for NRM: a dialog must occur between the terminal multiplexor and the authenticator in order to grant access.

Specification Implications: For the specification approaches which have been reviewed here, there are essentially two different ways of dealing with granularity of execution: either by showing the details explicitly, or by hiding the details via abstraction. For methods such as Hailpern-Owicki [Hail80], the protocol involved is formally specified and verified. In approaches such as Hoare's communicating sequential processes [Hoar85], Lamport's logic of actions [Lamp90], Lam and Shankar's theory of modules and interfaces ([Lam91a] and [Lam91b]), and the Abadi-Lamport composition approach [Abad90], provision is made for hiding internal states of a component. These two approaches are complementary, rather than mutually exclusive, and both may be useful for any given system.

3.0 Conclusion/Summary

The issues that we have identified here reveal a picture of composite systems as richly textured and complex structures. The specification approaches that we have surveyed vary with respect to such characteristics as type of component organization/composition (sequential and/or parallel), properties expressible (safety and/or liveness), and viewpoint (internal or external). Taken individually, each of the specification issues identified is addressable to some extent by one or more of the approaches that we have surveyed. However, no single approach addresses all of the issues that concern us.

Our efforts over the next year or two will be to develop a worked example of a composite system specification using automated tools augmented as necessary with additional (non-automated) formal methods. We anticipate that such an effort will necessitate a combination of an internal approach (for expressing such issues as element granularity, label consistency and component policies) with an external approach (for expressing component relationships, interactions and composition issues). Both sequential and parallel composition will be necessary, but with respect to properties we will focus on safety properties in order to provide some limit to the complexity of the system.

REFERENCES

[Abad90] Abadi, M. and Lamport, L. Composing Specifications. Research Report 66: Digital Systems Research Center, October 1990.

[Chen83] Chen, B. and Yeh, R. T. Formal Specification and Verification of Distributed Systems. IEEE Transactions on Software Engineering, Vol. SE-9, No. 6, November 1983.

[Fell87] Fellows, J., Hemenway, J., Kelem, N., and Romero, S., The Architecture of a Distributed Trusted Computing Base. Proceedings of the 10th National Computer Security Conference, 1987.

[Gamb90] Gambel, D., Walter, S., and Fordham, M. HSRP - A1'ing a Large-Scale Management Information System. Proceedings of the AFCEA Military/Government Computing Conference and Exposition, January 1990.

[Hail80] Hailpern, B. and Owicki, S. Verifying Network Protocols Using Temporal Logic. Proceedings of the IEEE Conference on Trends and Applications: 1980 Computer Network Protocols.

[Hoar85] Hoare, C.A.R., Communicating Sequential Processes. Prentice/Hall International, London, 1985.

[Kele91] Kelem, N. L. and Feiertag, R. J. A Separation Model for Virtual Machine Monitors. Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy.

[Lam91a] Lam, S., Shankar, A.U., and Woo, T. Applying a Theory of Modules and Interfaces to Security Verification. Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy.

[Lam91b] Lam, S. and Shankar, A. U. A Composition Theorem for Layered Systems. Proceedings of the 11th International IFIP WG6.1 Symposium on Protocol Specification, Testing, and Verification, Stockholm, June 1991.

[Lamp90] Lamport, L. A Temporal Logic of Actions. Research Report 57: Digital Systems Research Center, April 1990.

[Land84] Landwehr, C.E., Heitmeyer, C.L., and McLean, J. A Security Model for Military Message Systems. ACM Transactions on Computer Systems, Vol. 2, No. 3, August 1984.

[McCu88] McCullough, D. Noninterference and the Composibility of Security Properties. Proceedings of the 1988 IEEE Symposium on Security and Privacy.

[McLe88] McLean, J. The Algebra of Security. Proceedings of the 1988 IEEE Symposium on Security and Privacy.

[McLe90] McLean, J. The Specification and Modeling of Computer Security. Computer, January 1990.

[Scha84] Schaefer, M., and Schell, R. Toward an Understanding of Extensible Architectures for Evaluated Trusted Computer System Products. Proceedings of the 1984 IEEE Symposium on Security and Privacy.

[Shoc90] Shockley, W.R. Dockmaster TDI_Comments forum entry #221, August 24, 1990.

ISSUES TO CONSIDER WHEN USING EVALUATED PRODUCTS TO IMPLEMENT SECURE MISSION SYSTEMS

WILLIAM R. PRICE, LT COL, USAF[1]
Headquarters Air Force Space Command (LKXS)
Peterson Air Force Base, CO 80914

ABSTRACT

In spite of the availability of the Trusted Computer Security Evaluation Criteria and products evaluated against it, DOD organizations have fielded very few operational systems that effectively employ the offered security features. This paper examines some of the issues that builders need to consider when trying to develop secure systems.

1. Introduction

Although the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) [2], the "Orange Book," has existed for almost a decade, progress toward implementing systems based on it has been disappointing. Previous conferences have included reports of ongoing research and development projects attempting to demonstrate and extend computer security techniques in military scenarios. However, there have been few, if any, descriptions of secure operational systems implemented using either TCSEC evaluated products[2] or TCSEC based techniques. The premise is that secure operational systems do not exist in significant numbers. This paper examines why secure systems are not prevalent despite DOD policy based on the TCSEC and the availability of evaluated products.

This paper focuses on systems that directly support military operations. Key characteristics of the systems of interest are:

1. The major users are from operational (e.g., flight operations, aircraft maintenance, supply, air and space surveillance, satellite operations) rather than computer disciplines.

2. The systems support routine processing and flow of information. Users expect to enter and receive information in predefined formats.

3. User's have limited, if any, ability to create computer programs.

---

[1]The views expressed here are solely those of the author and not the Air Force.

[2]Systems employing TCSEC evaluated products but not using the security features are dismissed. Although many DOD systems employ TCSEC evaluated products, inquiries often reveal that systems operate without using crucial security features. A common indicator of such a situation is a computer system security officer of an alleged class C2 system who does not know what access control lists are or how the system uses them.

1.1. Objective

The point of the paper is that there has been too much reliance on over-simplified guidance. The approach frequently used to acquire "secure" systems has been to determine the desired TCSEC class and then reproduce or paraphrase the class description in the system specification. The Department of Defense established a methodology for determining the required TCSEC class based on the difference between the least cleared user's clearance and the most classified data's classification[3]. The approach apparently has not been successful. Simplistic methodologies are no substitute for thorough, quality system engineering.

1.2. Overview

The remainder of this paper examines some of the impediments that often interfere with attempts to implement secure operational systems. The categories of impediments discussed are ill-behaved problems, lack of requirements understanding, and design issues.

2. Ill-Behaved Problem Domains

There are environments where operational concepts are ill-behaved (i.e., inconsistent) with the concepts underlying the TCSEC. In most cases, the disconnects relate to the mandatory access controls. Mandatory controls work best in situations where data has easily determined, static security labels (classifications). There are situations where data dynamically changes security labels and where the only classified data results from the association of several pieces of information that are each individually unclassified. Examples of dynamically changing labels include scenarios where a planned event is classified prior to its occurrence but unclassified after it occurs. The date of a space launch may be classified prior to the launch but unclassified at the time of launch[4]. There are systems that produce classified products resulting from the association of data that is unclassified. Security classification guides occasionally identify such circumstances. There are guides that specify unclassified and classified associations where associations of A with B and B with C are each unclassified but A with C is classified. There are also situations where classified outputs result from unclassified inputs to an unclassified process.

Computer security advisors need to exercise caution in proposing TCSEC evaluated systems in these situations. Although the required analysis would point to a B level system, effective use of the security features, particularly the mandatory access controls, may be difficult. Situations involving dynamic changes to lower mandatory security labels may require manual intervention or "trusted" processes. TCSEC security features alone do not recognize the upward classification change that should result when data associations occur. Software must know, detect, and act when the associations occur. Evaluated product features that allow "write ups" and support incorporation of trusted processes would reduce development risks. Understanding the types and frequency of classification changes and the amount of data involved are prerequisites for making sound system design decisions.

---

[3]The methodology for determining a desired TCSEC class first appeared in the "Yellow Books" [4,5]. Subsequently, DOD Directive 5200.28 [1] and its Air Force implementation, AFR 205-16 [3], incorporated the methodology and, thereby, made use of the methodology mandatory.

[4]A space launch is often apparent to and observable by large segments of the population.

Before pursuing approaches needing B level evaluated products, one should explore other alternatives such as restructuring the environment to be better behaved or pursuing system high rather than multilevel mode approaches. In many cases the "direct" users have clearances for all data, but there is a desire to produce unclassified products for "indirect" users without manual review. Manual review or automated guard approaches may be more cost effective. In one case a system processing normally as unclassified had a requirement to perform classified processing for short periods. The system operated in a secure environment but had unclassified external interfaces. The system received classified information shortly before the occurrence of a classified event, but the information became unclassified after the event occurred. Rather than attempt multilevel operation, the solution was to operate in system high mode using periods processing. When the system received classified information, the facility entered classified operations. Operators severed unclassified external connections and did not release normally unclassified print products. Once the event occurred, the system returned to unclassified operations. There was no need for downtime to "sanitize" the system because the information was no longer classified.

3. Lack of Security Concept and Supporting Requirements

Early efforts to understand and define security requirements are essential. Simply copying or paraphrasing the requirements found in the Orange Book for the desired class is inadequate. Users of operational mission systems interact with mission specific software and not with the operating system. The set of security abstractions with which users deal is fundamentally different from the set provided by TCSEC evaluated operating systems. For example, users of an air defense system interact with displays showing aircraft in flight superimposed on maps, status of various air surveillance systems (e.g., radars), and the operational status of friendly air defense units. Users do not see (and do not want to know about) abstractions often considered important in TCSEC evaluated operating systems such as files, memory segments, interprocess communication mailboxes, and peripheral devices.

The system security requirements have to be consistent with an understanding from a security perspective of how users will employ the system to accomplish their mission (i.e., a security concept of operations) and an understanding of organizational and individual responsibilities for security of the operational system (i.e., a security policy). Ideally, accurate and up-to-date documents should describe the security concept of operations, the security policy, and the system security requirements. The documents should be consistent. Whenever one of the documents changes, there should be an assessment of the impact on the other two. The following paragraphs address the security concept, policy and requirements and illustrate some interactions among them.

3.1. Security Concept of Operations

The security concept of operations should identify the security issues and concerns related to the operation of the system and describe how to ensure security in the operation of the system. The concept should consider all security disciplines including both external (e.g., physical and personnel) and internal (e.g., hardware and software) measures. The security concept of operations should:

1. Describe the assumed physical environment in which the system operates.

2. Identify security relevant groups or communities of users. Users (individuals or groups) may have specific responsibilities and be expected to perform specific functions. Users include both "direct" users who interact with the system and "indirect"

users who may submit or receive information but have little or no ability to interact with the system. An indirect user may receive messages through "air gap" (e.g., printed reports or tapes) or direct electronic interfaces. Users include support personnel as well as those performing mission functions. The operations concept should include key support personnel such as hardware and software maintainers, computer operators, data base administrators, and system security officers (SSOs). A major issue is whether external or internal measures will monitor and control these personnel. Operational concepts for the security officer can have manning impacts. Particular issues are whether the security officer is a full-time function (only duty of one or more people) or part-time (performed by someone with other responsibilities) and whether a security officer will be present whenever the system operates, on-call, or only available during "day staff" operations. Operational considerations regarding expected or needed responses to attempted "hacker" penetrations or to the system "locking out" a legitimate user who forgot a password or exceeded the allowed number of logon attempts should guide these decisions on security officer manning.

3. Identify the data needed by and the functions required for each of the user communities as well as any data and functions which the system should deny to them.

4. Identify security significant events and data collections.

5. Specify the degree of need-to-know (including need-to-perform) controlled by internal measures. Of particular interest is determining the desire for need-to-know controls refined beyond any formal access determinations. For example, there may be information or functions that only certain individuals should perform (e.g., only the commander can transmit messages directing the use of force). External or internal measures could enforce such limitations.

6. Specify whether individual user identification and authentication and user accountability are necessary. Operational concepts may not bind specific users to specific terminals (e.g., large wall displays) or may require several individuals to use a single terminal. Operational communities may object to individual logons particularly at shift changes and want to substitute physical and administrative controls (e.g., crew schedules).

7. Identify events and the amount of information necessary for individual accountability purposes. For some security events, an audit trail entry indicating the event's occurrence may be sufficient. However, other circumstances may require recording specific details of the event (e.g., recording the "before" and "after" data for a write to a file).

8. Specify the length of time that audit records have to be retained and the frequency of their review.

If mandatory access controls are necessary, the concept of operations should also address:

1. The range of users' security clearances.

2. The range of data classifications.

3. The needed granularity of control over data.

4. The presence of information involving compartments, categories or special caveats.

5. Any situations where data dynamically change classification and the criteria for the change.

6. Operational impacts of users having to frequently change their security level because of prohibitions against "read up" and "write down."

The discussion above illustrated the need for consistency within and among the three documents. In several cases, tradeoffs between internal and external measures are possible. When the operational concept calls for internal measures, the security requirements must prescribe the desired internal measures. External controls should be consistent with internal controls. For example, users of systems supporting multinational defense organizations often include both United States and foreign personnel. Security controls intended to prevent disclosure of some information to foreign nationals are not particularly useful if a foreign national has an unobstructed view of a US user's workstation.

3.2. Security Requirements

The overall system specification generally includes the security requirements. The security specifications must describe the capabilities to support the concept of operations. Security requirement authors should not simply copy the appropriate Orange Book class description. An "Orange Book" class description can serve as guide, but substantial adaptation is necessary. The class descriptions contain abstract terms such as subject, object, security labels, and single-level and multilevel input/output devices. Thorough system engineering of the security requirements is essential. The security requirements should relate to other system requirements. These requirements, at a minimum, should replace Orange Book security terms with terms that relate to the rest of the system specification.

The security requirements should:

1. Identify the actions that are subject to either mandatory or discretionary access controls.

2. Identify the security objects to which the system must control access.

3. Describe the essential components of security labels and the range of values for each component.

4. Describe features for defining users and user groups.

5. Describe special measures for detecting and preventing "hackers" and unauthorized access attempts. Lock outs of terminals, network ports, or userids may seem appealing. However, there are tradeoffs. Such lock outs could cause denial of service. Simply reporting these events to the security officer or operator may be sufficient. Automatic logout after periods of non-use is a frequently advocated measure to prevent unauthorized access to unattended terminals. Any such requirements should address what should happen to ongoing processes associated with and subsequent messages destined for the terminal.

6. Describe the SSO capabilities and related resource requirements. The SSO's interface should be as "user friendly" as any other user's interface. Capabilities to support the SSO's responsibilities should be as detailed as those for any other user.

Examples of SSO capabilities are defining and managing the access rights of individual users, defining and managing group designations and membership, defining and changing security labels, unlocking userids or terminals that the system locked because users exceeded the allowed number of trials, and reviewing and analyzing audit trails. SSO resources might include terminals or workstations, storage space for audit trails, and processing capability to support audit trail review and analysis (particularly, if the operating concept calls for "off-line" review and analysis).

7. Describe security measures or capabilities associated with other "powerful" users (e.g., computer operators, maintainers, and data base administrators).

8. Identify performance requirements relative to the security features. If the system provides the capability to select which events to audit, the specification should identify the level of auditing in effect during general system performance testing. This level of auditing should correspond to that planned for normal operation. The storage capacity for audit data should be tested. If the requirements state that storage should be adequate for audit data collected over some period of time, the test should involve a representative "workload." Another performance issue may relate to the logon function, particularly if the system has critical time constraints. For example, if there is a mission requirement to respond to events within a few seconds of their occurrence, the operational community is unlikely to accept a few minutes as the time required for a user to log on. The operational community will include as part of logon any time required to load the application and initialize it prior to being ready to accept inquiries from the user.

3.3. Security Policy

The security policy marries the operational concept and the system specification. It should describe the roles, responsibilities, and duties of the individual system users and their organizations relative to the operation of the system. The policy should require and ensure the use of all external and internal measures necessary to satisfy the security concept of operations. All organizations affected should participate in its preparation and understand its impacts. Early agreement can prevent last minute surprises, organizational conflicts, and costly changes.

4. Design Issues

In a few circumstances there are products evaluated at the same class as that believed necessary for the desired system. Attempts to design an operational system that effectively uses the security features can be extremely challenging. Non-security issues may tempt system engineers to make tradeoffs that jeopardize the security of the system. Difficulties often encountered include the lack of available higher level, secure components (e.g., networks, data bases, windowing software, electronic mail), object granularity, performance, and auditing.

4.1. Lack of Secure Components

Development activities are under strong encouragement to develop systems using commercial off-the-shelf software (COTS). Unfortunately, much of this software is not compatible with evaluated products, particularly with mandatory access controls. For example, office automation and electronic mail systems may not work unless all users and information are at the same security level. Attempts to operate with differing security levels either cause errors or frustrate users because of the need to change security levels. For example, such systems often maintain a user specific directory or data base (e.g., a

directory of documents or messages) which must be stored in a security object of the evaluated product (e.g., a file). The component will not work if the user attempts to use the component while operating at a security level other than the level of the directory or data base.

Vendors are working on secure versions of many of these components, and some secure components are available. However, the components are usually specific to certain evaluated operating systems and sometimes are not easily integrated with the evaluated products.

4.2. Object Granularity

The common belief is that designers of secure applications software could map objects at the application level onto objects of the underlying evaluated product. The mapping could be one to one, one to many, or many to one, but the assumption was that mapped objects would all have the same security level. One difficulty is that the size of objects can be grossly mismatched. Implementing "small" application objects using evaluated product objects intended for "large" collections of data may impact performance or force developers to augment the evaluated product with "trusted software" that essentially implements a new type of security object.

4.3. Performance

The issues of granularity and performance are interrelated. As stated above granularity can cause performance problems. Another performance issue that has occurred with several systems relates to logons (i.e., user identification and authentication) in time critical systems. For some systems, time for one user to logout and another to logon at the same workstation (e.g., as might occur at a shift change) can take several minutes. Most of the time is for the application to load and initialize itself prior to being ready to accept input from the user. Often, the new user wants to resume exactly where the old user stopped. The process of destroying the old user's ongoing activity and rebuilding the same context for the new user seems unnecessary. A capability to change user accountability without disrupting basic mission processing is what the operational community desires. Unfortunately, none of the current evaluated products provide a rapid change of user accountability.

4.4. Auditing

Auditing needs to be addressed carefully. If the security objects at the applications level are not uniquely distinguished at the level of the evaluated product's objects, audit trail entries of the evaluated product may be useless. The applications software may have to provide its own audit trail of security events.

5. Conclusions

Several conclusions can be drawn:

1. Implementation of secure mission systems using evaluated products, while potentially feasible, is non-trivial.

2. Insufficient planning for the operations concept and security requirements is probably the major impediment to success.

3. The underlying security concepts of the Orange Book and techniques for implementing secure systems are not well-understood by many developers.

4. The lack of easily integrated, secure components adds complexity and technical risk to development efforts.

6. Recommendations

This paper has contained several detailed recommendations regarding the development of secure systems. Because of the conclusions above, priority should be given to identifying the security concept of operations and security requirements based on Orange Book guidance. In military applications involving limited security risks, implementation of the security controls in applications software rather than risky attempts to use an evaluated product may do more to advance the eventual implementation of secure systems. While this may be a controversial recommendation, the development of systems with proper functionality (but without full assurance) may do more to advance secure systems than continuing to wait for the "big bang" when a full set of secure components for building secure systems is available. Users and developers will gain knowledge and experience regarding the operation of secure systems.

7. References

1. Department of Defense, Department of Defense Directive (DODD) 5200.28, Security Requirements for Automated Information Systems (AISs), March 21, 1988.

2. Department of Defense, Department of Defense Standard: Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments, DOD 5200.28-STD, December 1985.

3. Department of the Air Force, Air Force Regulation 205-16: Computer Security Policy, 28 April 89.

4. DOD Computer Security Center, Computer Security Requirements -- Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments, CSC-STD-003-85, 25 June 1985.

5. DOD Computer Security Center, Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements -- Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments, CSC-STD-004-85, 25 June 1985.

The IT Security Evaluation Manual (ITSEM)

Y Klein [1], E Roche [2], F Taal [3], M Van Dulm [4]
U Van Essen [5], P Wolf [6], J Yates [7]

ABSTRACT

Three basic elements are required to carry out an IT Security Evaluation : Criteria for what has to be assessed; a Methodology for how the assessment is to be carried out; and a scheme to provide a framework in which such assessments may be conducted.

Four member states of the European Community (France, Germany, the Netherlands and the United Kingdom) first developed the Information Technology Security Evaluation Criteria (ITSEC), which have already had some success in attempting to meet the first requirement. This paper will review the recent development of the Information Technology Security Evaluation Manual [ITSEM], which presents a solution to the second requirement in the context of national schemes.

The paper will present an overview of the ITSEM and will then introduce and examine four fundamental principles of evaluation: Repeatability, Reproducibility, Objectivity and Impartiality. Fulfilment of these four principles is a technical prerequisite of international mutual recognition of the certificates which are issued to summarise the outcome of evaluation, and to confirm that they have been properly conducted.

The paper will also reflect the fuller consideration given in ITSEM to the Strength of Mechanism and Vulnerability Assessment concepts, previously introduced in the ITSEC, in order to illustrate how the fundamental principles are applied in practice.

INTRODUCTION

Four member states of the European Community (France, Germany, the Netherlands and the United Kingdom) developed the Information Technology Security Evaluation Criteria (ITSEC) to provide a basic set of criteria as a foundation for security evaluation.

However, in order to perform evaluations, as well as having criteria, it is necessary to have a methodology for evaluation, and an evaluation scheme to provide an organisational framework.

---

[1] Service Central de la Sécurité des Systèmes d'Information, Paris, France

[2] Department of Trade and Industry, London, United Kingdom

[3] Netherlands National Communications Security Agency, The Hague, The Netherlands

[4] Ministry of the Interior, The Hague, The Netherlands

[5] German Information Security Agency, Bonn, Germany

[6] Centre d'Électronique de l'Armement, Bruz, France

[7] Communications-Electronics Security Group, Cheltenham, United Kingdom

In order to establish an evaluation methodology, the four states have now produced the Information Technology Security Evaluation Manual (ITSEM). The objective of the ITSEM is to define methods for carrying out evaluations against the ITSEC in sufficient detail to ensure that an evaluation performed in one country can also be recognised by another country.

This aim of mutual recognition of evaluation results, leading to mutual recognition of certificates, is furthered in ITSEM through the definition of methods which promote the principles of repeatability, reproducibility, impartiality and objectivity.

This paper will provide a guide to the ITSEM document, an introduction to the four principles of repeatability, reproducibility, impartiality and objectivity, and then provide some illustrations of how the ITSEM applies these principles.

GUIDE TO ITSEM

The ITSEM is divided into 10 Chapters and 4 Annexes, and follows a similar presentation style to that of the ITSEC.

Chapter 0 presents an introduction to the ITSEM. Chapter 1 contains a short presentation of the scope of the ITSEM. Chapter 2 describes the Evaluation Process, defining the roles of the Certification Body (CB) and the Information Technology Security Evaluation Facilities (ITSEFs) and gives a framework for the organisational and procedural aspects to be followed during the conduct of an evaluation.

Chapter 3 explains the Evaluation Philosophy which underlies the ITSEC. It contains the principles which must be followed by the evaluators to achieve the aims of the ITSEC. It considers the principles of Feasibility of evaluation, Understanding by evaluators, and Mutual Recognition of evaluation results.

Chapter 4 contains the Evaluation Methodology. It describes the method as a means of generating evaluation work programmes, consistent with the evaluation philosophy and principles described in Chapter 3. The Chapter examines the tasks in sufficient detail for mutual recognition arrangements and provides objective instruction on how evaluators should identify vulnerabilities and undertake penetration testing. It considers the raison d'etre of security measures through an examination of assets, threats, risks and confidence. It explores the definitions of various ITSEC terms such as security objectives, components, functions and mechanisms. It draws the distinction between errors and vulnerabilities and presents a detailed account of vulnerability assessment.

Chapter 5 contains details on the contents and the scope of deliverables to be presented to an evaluation. This is more than just a consolidation of the deliverable requirements defined in the ITSEC. There are implicit as well as explicit requirements in the ITSEC. For example, the evaluators may need access to the operational site (in the case of a system evaluation) and may require informal training from the developers.

Chapter 6 contains details on the contents and the scope of the Evaluation Reports in order to be acceptable for consideration by a CB. It also gives guidance for the production of Certificates.

Chapter 7 identifies the different categories of tools and techniques which enable the preparation, the technical conduct and the administration of the set of tasks needed for the evaluation of the Target of Evaluation (TOE). It defines the major classes of use and specifies the qualities and characteristics to

which the tools have to conform. It cross refers each such category of tools to the various ITSEC criteria, level by level, and proposes the use of a 'Populated Integrated Project Support Environment' (PIPSE) in which such tools may be beneficially organised.

Chapter 8 explains when re-evaluation becomes necessary. A set of rules is stated and the consequences for a modified TOE are described. In particular the (basic) components of a TOE are categorised as being one of four types during initial evaluation. Four types of change are identified of varying substance and related to the four categories of component, level by level, in a set of tables indicating the consequence of such a change in terms of the need for re-evaluation.

Chapter 9 deals with the evaluation of TOEs which already contain evaluated components. It describes a model which enables the evaluator to identify those activities which have to be performed (or repeated) when composing pre-evaluated components into a new product or system.

Chapter 10 contains definitions of technical terms used within the ITSEM and provides full details of references made to external publications.

Annex A provides a definitive list of all forms of deliverable, level by level, as identified in Chapter 5. Annex B gives a detailed tabulation of the Completeness Criteria introduced in Chapter 3.

Annex C deals with the application of the ITSEC. It provides examples of the interpretation of various terms used in ITSEC, demonstrating how the ITSEC can be applied to the evaluation of systems and products. Seven examples are given spanning all ITSEC criteria.

A full index is provided to the document as Annex D.

FOUR FUNDAMENTAL PRINCIPLES

Evaluation has developed over the last ten years from a basic need to establish the level of confidence that can be placed in the security of a system or product. The enormous growth in the number of systems which require a measure of assured security has meant that the techniques of evaluation have spread worldwide. The danger inherent in this rapid growth is lack of standardisation in how evaluation is performed. The ITSEC is an attempt to provide a common set of criteria against which evaluation can be performed, and the ITSEM goes further, to develop a common framework for the evaluation process itself.

As with any process of testing something against some defined criteria, there are certain essential characteristics of the testing process itself which must be established. The most important of these are:

- Repeatability
- Reproducibility
- Impartiality
- Objectivity.

If evaluation adheres to these principles, then many advantages follow, both in the effectiveness of evaluation, and in the scope for mutual recognition of evaluation results.

Repeatability

This is defined as:

> repeatability: repeated evaluation of the same system or product (i.e. the same TOE in ITSEC terminology) to the same security target by the same ITSEF yields the same overall verdict (e.g. in ITSEC terms: E0 or [F-B3, E5]).

In order to ensure that this principle is applied, evaluators must follow established procedures. ITSEFs will therefore need to set up standard procedures for performing evaluations, and base them on well understood underlying principles. The ITSEM provides a definition of the evaluation process and describes its underlying philosophy, as well as providing guidance on generic evaluation work programmes as a framework for performing repeatable evaluation.

There is also guidance in the ITSEM on how to assign verdicts against the individual ITSEC criteria, which should assist in adhering to repeatability of the assignment of verdicts by different individuals within an ITSEF.

Reproducibility

This is defined as:

> reproducibility: evaluation of the same TOE to the same security target by a different ITSEF yields the same overall result as the first ITSEF (e.g. E0 or [F-B3, E5]).

This principle is similar to repeatability, but is more difficult to achieve, as it places extra constraints on the commonality of approach between different ITSEFs. To apply this principle there has to be a level of organisation for evaluations which ensures that all ITSEFs are operating in a common way, which allows their evaluation verdicts to be comparable. The ITSEM lays down some basic rules for CBs and national evaluation and certification schemes to provide a framework for reproducibility between ITSEFs in one country. The aim of reproducibility of results from ITSEFs in different countries is being pursued by groups discussing mutual recognition of evaluation results.

Impartiality

This is defined as:

> impartiality: evaluation is free from unfair bias towards achieving any particular result.

This is primarily a concern of CBs, who must ensure that no commercial or personal bias is introduced, through relationships between developers or sponsors of TOEs and the ITSEFs performing the evaluation. The ITSEM provides a framework of rules for who may (and may not) perform evaluations, which is aimed at ensuring that such conflicts of interest do not occur. It also establishes a need for reviews of the conclusions reached by an evaluator, by others within the ITSEF, with the objective of eliminating any individual bias in the results.

Objectivity

This is defined as:

> objectivity: a property of a test whereby the result is obtained with the minimum of subjective judgement or opinion.

In the past, the results of evaluation have depended to a large extent on the individual knowledge and experience of evaluators. The technique of evaluation has to be based on individual evaluators investigating a TOE until they have established a degree of confidence in its security. As well as being very subjective, this approach has also worked against the principles of repeatability, reproducibility and impartiality.

Obtaining completely objective results is an unattainable goal, however the ITSEM provides a number of means by which objectivity can be improved.

The most significant of these is the framework of a set of detailed evaluator activities based on the ITSEC, together with guidance on how these activities should be carried out. The ITSEM provides guidance on how an evaluator should assign verdicts against the individual ITSEC criteria, based solely on the evidence provided by the sponsor. In this way, opportunities for subjective judgements are reduced.

Guidance is also provided on how far an evaluator should go in investigating the TOE through the provision of Completeness Criteria; corresponding metrics are derived by the evaluator, to show that he has performed all the evaluation checks that are required and no more. The principle of independence of evaluators is introduced to ensure that the temptation simply to agree with the sponsor's evidence is removed by the need to perform independent analysis.

All the above techniques are introduced, through the ITSEM, to try to increase the objectivity of evaluation, to improve the process technically, but also to provide a foundation for mutual recognition of evaluation results.

Mutual Recognition

Security evaluation is one of many activities for which international mutual recognition is sought. ISO Guide 25 and EN45001 lay down guidelines for creating a framework for objective testing, regarding all types of products whether IT related or not in order to ensure that international mutual recognition of test results is possible. In the UK, the National Measurement Accreditation Service (NAMAS) has produced NIS35 [NIS35], a specific interpretation of the general regulations for IT testing.

Mutual recognition has been achieved in other fields of testing by a process of accreditation, i.e. by agreeing to recognise the technical competence and the impartiality of a test laboratory. International and European Standards (ISO Guide 25 [GUI25] and EN45001 [EN45]) have been established to provide guidance for this purpose, and the Western European Laboratory Accreditation Cooperation (WELAC) had been set up to implement them. Clearly, it was sensible to utilise this approach in the achievement

of international mutual recognition of security evaluation and certification. Thus, the Four Nations decided that evaluation laboratories should be accredited under a recognised Accreditation Scheme as a condition of licence to perform evaluations against the ITSEC. In order to make this possible, it was decided not only to harmonise existing evaluation methods but to ensure their compliance with ISO Guide 25 and EN45001. Such a strategy should form a sound technical basis from which international mutual recognition agreements can emerge.

The principal aim of the ITSEM is therefore to present sufficient detail of evaluation methods and procedures to ensure that technical equivalence of evaluations can be demonstrated. The intention, however, is not to unduly constrain the implementation of National Schemes and thus the scope of the ITSEM is limited to just that sufficient to allow demonstration of technical equivalence. However substantial information is included to ensure that the principles of repeatability, reproducibility, objectivity and impartiality can be fulfilled.

IMPLEMENTATION OF THE PRINCIPLES IN ITSEM

The above section introduced a number of areas in which the ITSEM has extended the basic concepts in the ITSEC to embrace the principles of repeatability, reproducibility, impartiality and objectivity, with the objective of improving the evaluation process, and obtaining mutual recognition.

The ITSEM does this by:

- providing a definition of the evaluation process and the roles of the participants
- describing a technique and procedural framework for reviewing evaluation results
- defining a common philosophy of evaluation to be applied by all participants
- providing a framework for evaluation methods and evaluation work programmes
- defining a consistent and objective technique for assigning verdicts
- describing a technique by which completeness of the evaluation requirements can be demonstrated
- providing procedures for performing vulnerability assessment
- defining a way in which security mechanisms can be assessed for strength
- providing outline standards for the form of documents recording the results of evaluation
- establishing a framework for decisions on the Re-evaluation of TOEs and the Re-use of evaluation results.

The two areas of Strength of Mechanisms and Construction Vulnerability Assessment will be considered further below, to provide some examples of how the ITSEM promotes adherence to the four principles identified earlier.

Strength of Mechanisms

A number of correspondents in commenting on the ITSEC pointed out that the measurement of strength (resistance to direct attack) as given in the ITSEC is subjective. For this reason the ITSEM elaborates on the ITSEC to give a more objective account of such measurement.

In simple terms, the ITSEM introduces four factors, each with just three values (or value ranges) which could be determined experimentally, at least in principle. They are:

- TIME: the time taken to make a successful attack (including all failed attempts)
- COLLUSION: the necessary 'inside' assistance required to prepare or actually make the attack
- EXPERTISE: the minimum expertise required to carry out the attack
- EQUIPMENT: the equipment required to make the attack.

Two heuristic tables are provided, one relating TIME and COLLUSION and the other EXPERTISE and EQUIPMENT. The evaluator determines the values of each of these four factors and simply adds together the two numbers found by looking up TIME and COLLUSION in the first table and EXPERTISE and EQUIPMENT in the other. The ITSEM then relates the results of this addition to the scale basic, medium and high.

Strength of Mechanism scores	
Score	Strength
1	not even basic
2 - 12	basic
13 - 24	medium
>24	high

For example, if a countermeasure can be defeated 'within minutes' 'alone' (TIME*COLLUSION = 0) by a 'layman' 'unaided' (EXPERTISE*EQUIPMENT = 1), then the overall score is 1, which the ITSEM tells us is not even basic. If the countermeasure can be defeated 'within days' with the necessary assistance of (another) authorised 'user' (TIME*COLLUSION = 12) by a 'proficient' attacker 'using domestic equipment' (EXPERTISE*EQUIPMENT = 4) then the overall score is 16, which is medium. The ITSEM defines the permitted values of the four factors, for example 'proficient' means "a person thoroughly familiar with the internal workings of the TOE but inexperienced with the workings of the underlying principles and algorithms of the type or actual security mechanisms involved".

It can be seen that while this does not provide a completely objective measure of strength, it provides a result which depends on a number of easily defined factors. The resultant rating of strength will consequently be based on a more objective assessment than one which consisted of a single subjective judgement.

Construction Vulnerability Assessment

The ITSEC identifies this effectiveness criterion, and gives a brief description of the evaluator actions that should be performed. Two of the areas it identifies where the evaluator should perform analysis are:

- perform independent vulnerability analysis
- perform penetration testing.

These two areas are expanded in the ITSEM, to provide clarification of the meaning of the terms, and to provide guidance on how these actions should be performed.

In order to perform an independent vulnerability assessment the evaluator must:

- take each representation in turn (starting with the architectural design and proceeding with the detailed design and implementation as determined by the evaluation level in question) and, following the determination that each representation is a correct refinement of its corresponding higher level representation
- determine whether or not an attacker could use the information present in each representation to defeat the objective of a countermeasure. (The countermeasures and their objectives are identified in the security target.)

An empirical procedure for identifying construction vulnerabilities, in the form of a (non-exhaustive) checklist, is to determine whether any of the following can be used to defeat the objective of a countermeasure:

- change the predefined sequence of invocations of components (as defined at this level)
- inject the execution of a component into the predefined sequence (also execution of data can be injected)
- use interrupts or scheduling functions to disrupt timing
- directly access (read, modify) internal data (secrets, local variables)
- indirectly access internal data (secrets, local variable)
- execute data not intended to be executed or make them executable
- use a component in a different context or give it a different semantics
- make use of new data objects introduced at this level
- disrupt concurrency
- use interference between components which is not visible at a higher level of abstraction

- invalidate assumptions and properties which are to remain valid at lower levels of abstraction.

If this checklist is consistently applied by all evaluators it should provide an improvement in objectivity of assessment, as well as gains in repeatability and reproducibility.

The ITSEM's guidance on the evaluator activity Perform Penetration Tests is given in the form of a set of procedures on how the work should be organised.

Penetration testing is defined in the ITSEM as the activity of:

- understanding the structure of the TOE and the security target
- generating hypotheses about flaws, i.e. finding potential vulnerabilities
- confirming the presence of those flaws, i.e. demonstrating, using penetration tests, that the potential vulnerabilities are, or are not, exploitable is practice
- generalising the flaws, i.e. considering whether the exploitable vulnerabilities are symptomatic of deeper vulnerabilities in the system.

In order to perform penetration testing the evaluator should therefore carry out the five sub-activities of:

- Prepare For Penetration Tests
- Identify Penetration Tests
- Specify Penetration Tests
- Execute Penetration Tests
- Follow-up Penetration Tests.

Prepare For Penetration Tests consists of ensuring that all parties concerned (developer, sponsor, user etc) prepare well in advance for provision of access to the TOE and other facilities needed to perform tests.

Identify Penetration Tests is gathering together all the information on potential vulnerabilities that has been collected in other evaluator actions, particularly other actions under Construction Vulnerability Assessment. The techniques of Completeness Criteria are used to ensure that all vulnerabilities, and their modes of exploitation, have been considered.

Specify Penetration Tests is the production of a complete specification of all the penetration tests that are to be performed, and a test plan for their execution.

Execute Penetration Tests is the performance of the defined tests, as determined in the test plan.

Follow-up Penetration Tests involves the activities associated with recording the results of tests, informing appropriate parties of any findings, and dealing with any consequences of the results.

Ensuring that penetration testing is always carried out in a similar way in all evaluations will help to ensure repeatability and reproducibility, and through the use of consistent techniques, the aims of objectivity and impartiality will be supported.

These examples show how the ITSEM promotes these four principles through the use of evaluation techniques, in the areas of technical analysis methods, the use of checklists and the following of defined procedures.

THE WAY AHEAD

There are a number of areas in which the ITSEM, at Version 0.2, is known to fall short of providing all that is needed in guidance for objective evaluations. A workshop is planned for September 1992, at which experts in the field will be invited to make contributions to the development of objective techniques and to the next issue of the ITSEM.

It is expected that this input will result in production of a further version of the ITSEM early in 1993, which should be usable for a trial period, in the same way as the ITSEC Version 1.2 has been on trial.

At the end of the ITSEC's trial period a new version will be published, to take account of the ITSEM and the practical experience that has been gained through the use of the ITSEC in evaluation.

The main outcome from the establishment of the ITSEM as an international standard will be the framework it provides for mutual recognition of evaluation results, based on the principles of repeatability, reproducibility, impartiality and objectivity. There is much work to do in establishing mutual recognition agreements based on the ITSEM, and in setting up national schemes which conform to its requirements.

REFERENCES

EN45 General Criteria For The Operating Of Testing Laboratories, CEN/CENELEC, June 1989

GUI25 ISO Guide 25, General Requirements For The Technical Competence Of Testing Laboratories, International Standards Organisation, 1982

NIS35 Interpretation Of Accreditation Requirements For IT Test Laboratories For Software And Communications Testing Services, NAMAS Information Sheet NIS 35, NAMAS Executive, National Physical Laboratory, United Kingdom, November 1990

ITSEC Information Technology Security Evaluation Criteria (ITSEC), Harmonised Criteria Of France, Germany, the Netherlands, United Kingdom, Version 1.2, June 1991, published by the Commission of the European Communities, ISBN 92-826-3004-8

ITSEM Information Technology Security Evaluation Manual (ITSEM), Version 0.2, April 1992, published by the Commision of the European Communities.

THE KINETIC PROTECTION DEVICE

Gregory Mayhew, Richard Frazee and Mark Bianco

Hughes Aircraft Company Ground Systems Group
P.O. Box 3310, Building 600 Station F241
Fullerton, California 92634

Abstract

The two general categories of symmetric key cryptographic algorithms are block ciphers and stream ciphers. Stream ciphers have inherently less encryption and decryption latency than block ciphers. However, stream ciphers are more difficult to design than block ciphers because the properties of the keystream must be carefully controlled. Hence in the public domain, the available devices are predominantly based on block cipher algorithms. Hughes Aircraft Company Ground Systems Group has developed a digital stream cipher for which the algorithm produces keystream satisfying all appropriate measures of randomness. The implementation is capable of megabits per second operation. The breadboard design is being transferred into an application specific integrated circuit by Hughes Aircraft Research Laboratory as a demonstration vehicle for reverse engineering protection of integrated circuits.

Keywords: stream cipher, binary sequences

Problem Statement

In a "classical" or symmetrical key system, cryptographic security can be maintained only if the key is held in private. Because only authorized users have access to the key, secrecy and authentication are provided at the same time. Assuming the secrecy of the key is maintained, then the security depends on the strength of the cryptographic algorithm. The two general categories of symmetric key cryptographic algorithms are block ciphers and stream ciphers. In block ciphers, the information stream is first segmented and then each segment is put through a series of invertible permutations and substitutions. In stream ciphers, a pseudo random bit stream is added bit by bit (Exclusive ORed) to the data stream. Stream ciphers have inherently less encryption and decryption latency than block ciphers. However, stream ciphers are more difficult to design than block ciphers because the properties of the keystream must be carefully controlled [1]. Hence in the public domain, the available digital devices for "classical" systems are predominantly based on block cipher algorithms. Examples of digital block ciphers are the Data Encryption Standard, the FEAL-N, and Teledyne Electronics Dynamic Substitution Device [2, 3].

Hughes Aircraft Company Ground Systems Group has developed a digital stream cipher for which the algorithm produces keystream satisfying all appropriate measures of randomness and for which the implementation is capable of mega bits per second operation. The Kinetic Protection Device (KPD) was originally conceived as a replacement module for the Department of Defense (DoD) cryptographic device embedded in the Position Location and Reporting System (PLRS) units and Automatic Location and Data Networking System (ALADNS) units. The KPD has been subjected to extensive standard statistical tests for pseudo randomness and the results are presented. The KPD breadboard design is in the process of being transferred into an Application Specific Integrated Circuit (ASIC) by Hughes Aircraft Company Research Laboratory as a demonstration vehicle for reverse engineering protection of integrated circuits.

PLRS and ALADNS

PLRS provides the basic tactical functions of position location, navigation, identification, and digital data communication in a hostile ground environment [4]. A PLRS community consists of one Master Station (MS) and about 400 User Units (UU). The master station and the user units are modems participating in a time division multiple access network. Each user transmits in its assigned time slots. The transmissions are in the 420 to 450 MHz UHF band. The anti-jam features are short burst transmissions, about 5 MHz of direct sequence spreading on each transmission, adjustable transmission power up to about 100 Watts, and forward error correction on all transmissions. All units perform time of arrival measurements on any transmission that they are able to receive. The time of arrival measurements are then reported to the MS through a hierarchy of relays. The master station performs multi-lateration on all the ranging measurements to obtain everyone's relative position. This position information is then used to provide navigation information to all users and to provide a location display in real time of all users within the system's operational area to the master station personnel.

A user unit can be configured as a manpack, vehicular, or rotary wing airborne unit. The militarized unit weights 13 pounds. The manpack requires one lithium battery which weighs 3 pounds. The user unit consists of an RF section, a signal processor, a secure data unit, a message processor, a barometric section, and an operator interface. The message processor converts operator requests to system messages or composes routine system messages. One such system message is the barometer reading so that unit height above sea level can be factored into the multilateration equations. Every message is encrypted by the digital secure data unit and forward error correction encoded by the signal processor. The secure data unit prevents unauthorized users from gaining access to the network and prevents repeat jammers from spoofing the network.

The master station has all the modem components of the user unit contained in a command response unit as well as a suite of 3 militarized computers and a 19 inch circular graphics display station. The computers perform all the network management, message traffic control, position location, unit tracking, and generation of operator displays. The MS is inside a shelter which is transportable by a 5 ton truck.

ALADNS is essentially small community PLRS. By reducing the maximum number of units to 64, the individual users data rate can be increased correspondingly. A more powerful microprocessor was also incorporated into each user unit so that the network management and position location computations are now distributed uniformly amongst all users in the network.

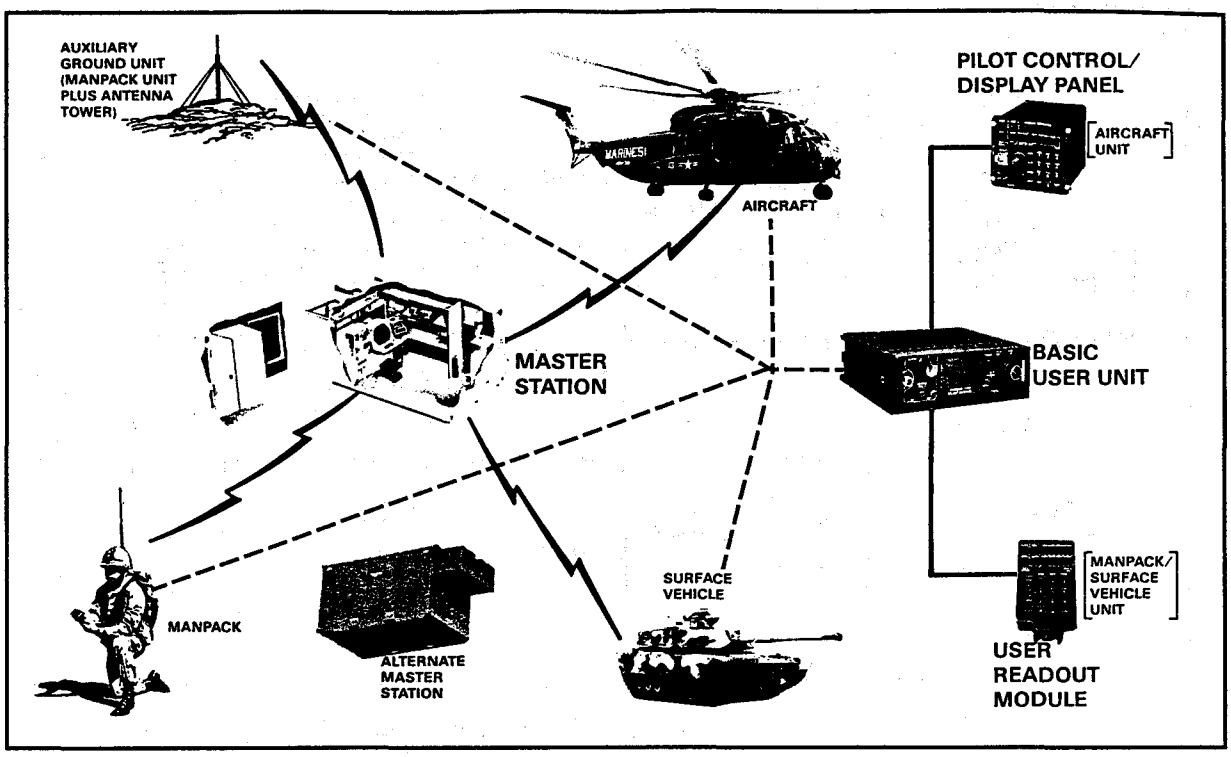


Figure 1. PLRS Network Components

KPD Architecture

The digital secure unit is inherent to the operation of PLRS and ALADNS. However, in general DoD cryptographic devices cannot be exported. For foreign sales of PLRS or ALADNS, an alternative, exportable cryptographic device was required. The device must duplicate the system interfaces exactly and must provide sufficient cryptographic strength. Figure 2 illustrates the functional areas of the KPD.

The master controller interprets the basic commands from the host unit. With the proper signalling, the control function will interpret the information on the data lines as plaintext or ciphertext. The control function then interacts with the keystream generator and the message validation functions to orchestrate the digital data through all the proper steps.

The crypto function includes message ciphering and message validation. Prior to message encryption, 10 bits of message validation are computed for each message. The message validation bits are protected by the encryption so that false messages cannot be inserted into the system. After message reception and decryption, message validation bits are computed on the received data bits. The computed validation bits are compared with the received validation bits and the result is reported.

The message validation algorithm is a cyclic redundancy check [5]. The CRC for the KPD uses the polynomial $X^{10} \oplus X^3 \oplus X^2 \oplus X^1$. With this polynomial, the message validation detects all odd number of errors, detects all double errors, detects all burst errors of length ≤ 10, and detects 99.8 % of all longer burst errors.

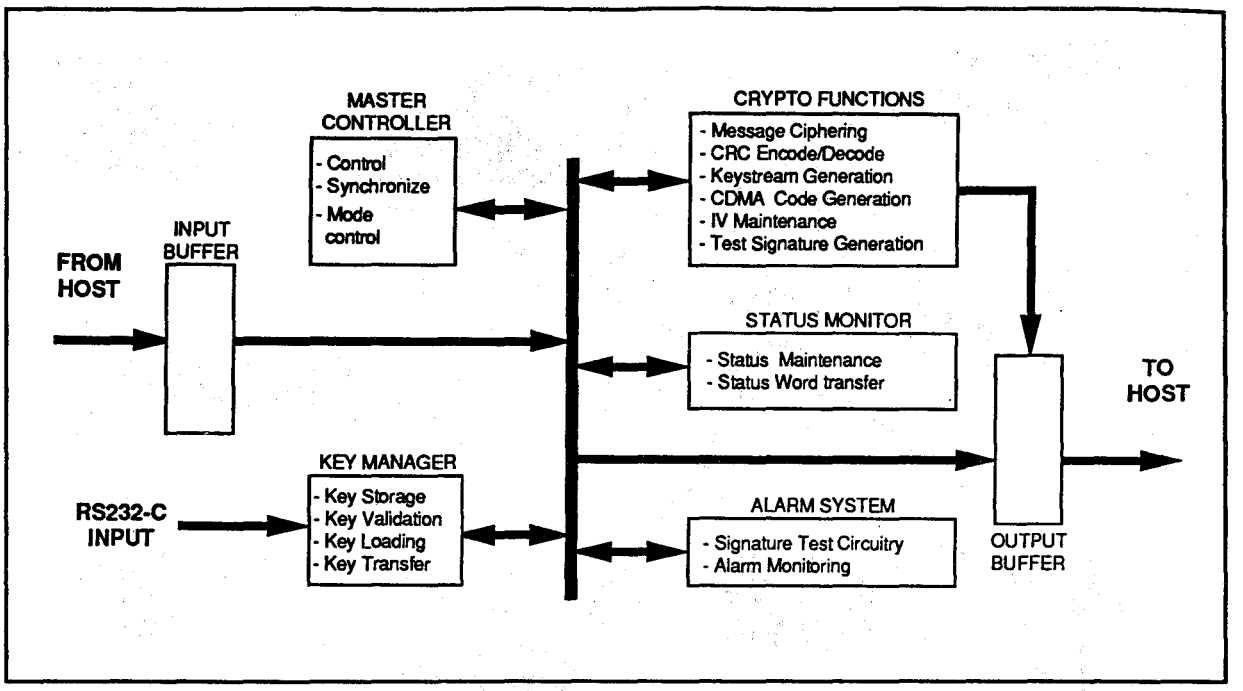


Figure 2. Kinetic Protection Device Functional Diagram

The keystream generator is a Key Auto Key stream cipher [6]. An initializing vector (IV) is used to produce the initial bit of the cryptographic keystream. At all subsequent time in the enciphering or deciphering operations, the IV is modified using keystream rather than plaintext or ciphertext. The advantage of this approach is that each bit error in the ciphertext induced by channel noise affects only that corresponding plaintext. Hence the ciphering process does not create error extension. A cryptographically strong stream cipher must not reoriginate, so the IV is modified prior to the encryption of any message. Each participant in the network must update their IV at any network transmission opportunity even if that participant does not have a message to transmit in any given network transmission opportunity.

The KPD includes cryptovariable key storage for the current data protection key, the next data protection key, and a special key protection key. Thus with one active data protection key the KPD support one level of data security. With the key protection key the KPD provides over the air rekeying (OTAR). The KPD also performs status monitoring and fault checking.

KPD Keystream Generator Design

The only unbreakable cipher is one with a keystream that never repeats and contains neither meaning nor pattern. Such a system is the one time pad. In a one time pad, random numbers from a printed sheet are added to the number value of each letter of a message. With high rate communications it is impossible in a practical sense to store a sufficiently large "pad" at the participants. Rather, a one time pad is emulated by algebraically generating sequences with various randomness properties. Any sequence generated algebraically will eventually repeat so the sequences are pseudo-random or pseudo-noise (PN) instead of truly random. Stream ciphers are an attempt to digitally duplicate the properties of the one time pad.

Stream ciphers can be categorized according to the mechanics of generating the keystream. Typical methods involve lookup tables, noise generating devices (diodes), or shift registers. Methods which result from lookup tables and noise generating devices are quite cumbersome. On the other hand, shift registers generate sequences which look random in every sense yet are easy to construct algebraically. This shift register property enables a cooperative transmitter and receiver to easily comprehend messages which appear only as noise to an interceptor. The KPD is a stream cipher which has a shift register as its foundation. The mathematical properties of shift registers were considered to find a cryptographically acceptable design and the hardware issues of shift register implementation were considered to select a realizable design.

As shown in Figure 3, a shift register consists of the memory assembly which acts as a state machine, a feedback function which determines the basic operation, and an output function which interfaces to the external world.

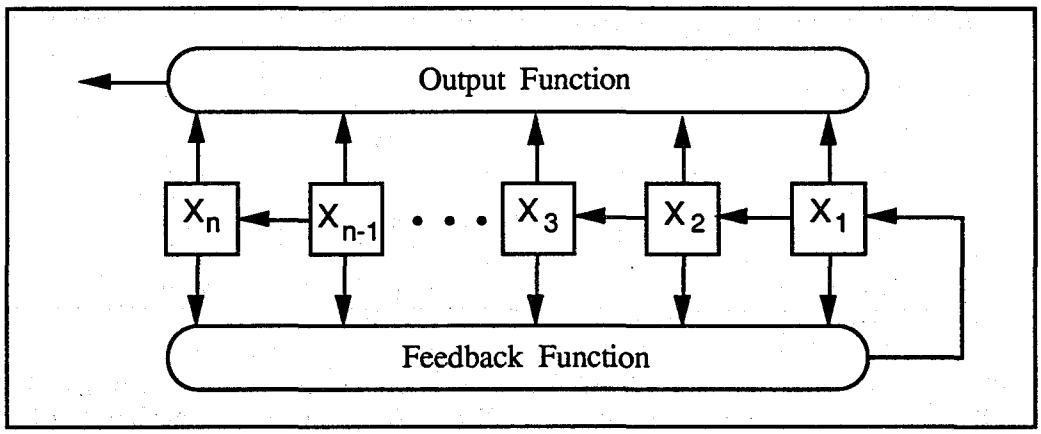


Figure 3. Generic Shift Register Structure

The goal of the feedback function is to put the n stage shift register through all 2^n-1 non-zero states before the states repeat. The feedback function can be linear or nonlinear. The full period linear functions produce M sequences and the full period nonlinear functions produce de Bruijn sequences [7]. The process of selecting full period feedback functions is well understood for linear feedback but not for nonlinear feedback. Similarly, the output function can be linear or nonlinear. Linear output functions do not inject any randomness properties into the keystream whereas nonlinear output functions do inject randomness properties into the keystream.

One method to upper bound the cryptographic strength of a stream cipher is to determine the linear span of the keystream [8]. The *linear span* of a sequence is the least order recursion relationship with binary coefficients that can duplicate the given sequence. This corresponds to a shift register with a linear feedback function and a single tap linear output function. If a sequence has linear span L, then after 2L successive elements of the sequence are known, the remainder of the sequence can be exactly predicted. As n grows large, 2n is an extremely small portion of the 2^n-1 period of a linear feedback function. To get acceptable performance, a linear feedback function can be combined with a nonlinear output function. The linear span L of the resulting sequence is then a function of the shift register length n and the degree of the nonlinear output r [9].

$$L = \sum_{i=1}^{r} \binom{n}{i} \quad \text{where } \binom{n}{i} \text{ is given by } \frac{n!}{i!\ (n-i)!}$$

As shown in Table 1 for a 13 stage shift register, the combination of linear feedback and nonlinear output rapidly approaches the cryptographic strength of nonlinear feedback alone. The difference is that, even for modest shift register sizes (20 stages), nonlinear feedback cannot be accomplished whereas linear feedback and nonlinear output can easily accomplished.

Table 1. Effect of Nonlinear Output on Linear Span

Feedback Function	Degree Output Function	Predictable after # bits *	% of period needed *
Linear	1	26	0.3
"	2	182	2.2
"	3	754	9.2
"	4	2184	26.7
"	5	4758	58.1
Nonlinear	1	> 4096	> 50

The combination of linear feedback and nonlinear output is the basis of the KPD design. The KPD block diagram is shown in Figure 4. The KPD is a digital stream cipher which produces 1 bit of output keystream for every clock pulse. The shift register size is 61 stages, the feedback is any valid linear feedback function, and the nonlinear output function is variably selected from a set of degree 4 functions. The shift register size was chosen for two reasons. First, the full period cycle should be significantly longer than any portion of the cycle that will be used. Assuming continuous operation at 1 GHz, a 61 stage shift register has a period length of 1 century. Second, the linear feedback functions for a 61 stage shift register are easier to select. When the number of stages corresponds to a Mersenne exponent — 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, and 127 — the irreducible polynomials are also primitive polynomials so less testing is required by the designer but not by the interceptor to select valid feedback functions [10].

The key for the KPD is 64 bits. Of these 64 key bits, 60 bits perform the selection of the feedback taps. For the feedback portion of the key, the size of the key space is 2^{60} or 1,152,921,504,606,846,976. The number of valid keys is 37,800,705,069,372,032 or $(2^{61}-2)/61$. A dedicated high end personal computer or minicomputer is required to perform the necessary key generation function. The remaining 4 key bits are to select 1 of 16 nonlinear output functions. These nonlinear functions are stored in ROM. The implementation of the nonlinear output functions in ROM enables the system to be changed in the event of compromise without scrapping the entire security system.

At present, the nonlinear output functions in the KPD have degree 4. This corresponds to a theoretically estimated linear span of 559,736. Therefore, a minimal sample of 1,119,472 consecutive bits should be necessary in order to possibly compromise the KPD design. The

KPD was designed for an environment in which at most 200 consecutive bits would potentially be available for compromise. Obviously, a linear span of 559,736 enables the KPD to cipher considerably longer messages than 200 bits.

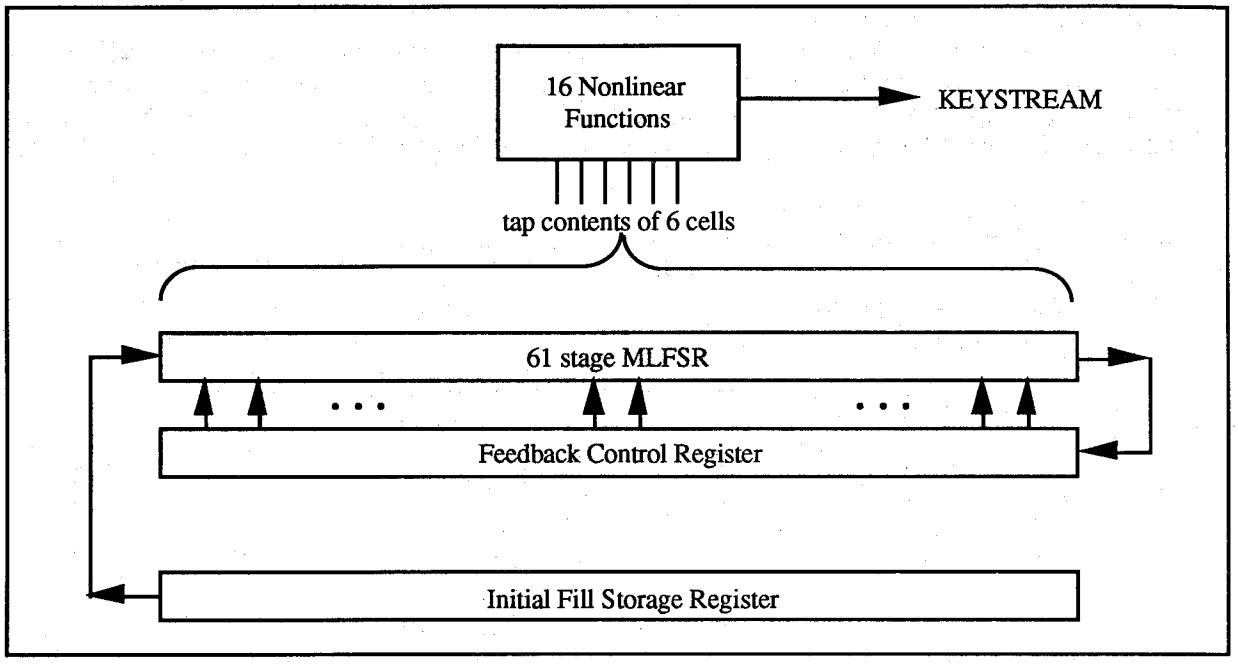


Figure 4. KPD Keystream Generator Block Diagram

KPD Keystream Statistical Analysis

The randomness characteristics of the KPD keystream will depend critically on the functions used in the ROM. Furthermore, the majority of these properties must be determined by testing rather than by analysis. The randomness properties to be tested are the balance property, the first delta property, the second delta property, the third delta property, and the polybit property [11, 12]. The remaining randomness property, the linear span of the keystream, can be estimated by analysis and then verified by testing.

The keystream produced by the KPD has been evaluated using extensive standard measures of statistical randomness. Values quoted for each of the randomness properties correspond to a perfectly random stream. A design is evaluated by how close the pseudo-random keystream approaches the perfectly random keystream. Minimum sample size to evaluate a design is 1 million bits.

The *balance property* is satisfied if the total number of ones in the sample divided by the sample length is 0.5. The KPD balance value is 0.5001. The *first delta property* is satisfied if the total number of overlapping 00 and 11 patterns divided by the sample length is 0.5. The KPD first delta value is 0.4993. The *second delta property* is satisfied if the total number of overlapping 0ø0 and 1ø1 patterns divided by the sample length is 0.5 (ø is don't care.) The KPD second delta value is 0.5004. The *third delta property* is satisfied if the total number of overlapping 0øø0 and 1øø1 patterns divided by the sample length is 0.5 (ø is don't care.) The

KPD third delta value is 0.4993. The *polybit property* is satisfied if there is a uniform distribution of the 2^m possibilities on length m non-overlapping segments of the keystream. For the sample size of the KPD evaluation, the expected polybit value is 488 and the KPD Chi square value was 615. Therefore, according to all the statistical tests performed, the KPD is producing high quality pseudo-random keystream.

With a shift register length of 61 and nonlinear output functions of degree 4, the theoretical estimate of the linear span for the KPD keystream is 559,736. Random samples of the keystream have had verified linears spans in excess of 100,000 bits.

For applications requiring larger linear spans, the KPD architecture provides a simple mechanism for increasing the cryptographic strength of the keystream. For example, a degree 6 output function will result in a keystream with a linear span of 62,034,255 and will require a minimum sample of 124,068,510 consecutive bits to compromise. Similarly, degree 7 and degree 8 output functions will result in a keystream with linear spans of 442,779,663 and 3,387,607,428, respectively. So degree 7 and degree 8 output functions will require a minimum sample of 885,559,326 and 6,775,214,856 consecutive bits, respectively, to compromise. Computing linear spans on 30,000 bits of keystream typically takes 6 hours of CPU time on a 1 MIP machine. Thus, any enhanced KPD design is reasonably beyond the capabilities of the majority of potential interceptors.

KPD Breadboard and ASIC

The KPD unit consists of a master controller, a key manager, a cryptographic functions section, a status monitoring section, an alarm system, and input/output buffers. The KPD breadboard is a 9 inch by 9 inch wire-wrap board. The KPD breadboard design uses 72 integrated circuits. The master controller and key manager utilize the Altera erasable programmable stand alone microsequencers (EP-SAMs). The EP-SAM is a highly versatile microsequencer which has re-programmable microcode. The microsequencers are easily re-programmable to add functionality or to change interface timing. Another 39 integrated circuits are low density Altera erasable programmable logic devices (EPLDs). The EPLDs provided a high degree of flexibility during design, test, and debug. Much higher density EPLDs (~ 5:1) and programmable gate arrays (Xilinx) are now available. The remaining 31 integrated circuits are all high speed complementary metal oxide semiconductors (HCMOS). The core keystream algorithm requires only 12 integrated circuits. The majority of the integrated circuits are for key management, fault testing, and interface timing.

The KPD is a digital, stream cipher which produces 1 bit of output keystream for every clock pulse. The operating speed of the KPD is determined by the implementation technology rather than the algorithm. The KPD is presently designed to operate at up to 16 MHz clock rate. The operating speed of the present KPD breadboard is currently limited by the operating speeds of the older Altera EPLDs. Replacing the older EPLDs with newer EPLDs will increase the breadboard operating speed. HCMOS circuitry on the KPD breadboard will accommodate up to a 25 MHz clock. Because the KPD design is all digital, faster operating speeds are obviously possible with the appropriate technology, such as ACMOS or gallium arsenide.

Hughes Research Laboratory in Malibu, California is currently transferring the KPD breadboard design into a custom Application Specific Integrated Circuit. Hughes Research Laboratory is using the KPD ASIC as a means to demonstrate its patented Design Protection and Usage Control (DP&UC) process for integrated circuits [13].

The first portion of the DP&UC, design protection, is a collection of active and passive techniques to prevent reverse engineering of integrated circuits. A integrated circuit with these design protection techniques looks topologically different than it operates, even when focused ion beam, infrared inspection, and planer slicing reverse engineering processes are applied. The design protection also includes circuit mask protection. Thus someone using an unauthorized copy of the mask will produce a malfunctioning integrated circuit.

The second portion of the DP&UC, usage control, is the incorporation of authorization keys into the integrated circuit design. Thus, either an authorized copy without a proper key or an unauthorized copy of the integrated circuit will function differently than an authorized copy of the integrated circuit with a proper key. The key storage portion of the integrated circuit is also protected against reverse engineering.

References

1. Rueppel, Analysis and design of stream ciphers, Springer Verlag, Berlin, 1986.

2. Data Encryption Standard, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, Washington, D.C., January 1977.

3. Murphy, "The Cryptanalysis of FEAL-4 with 20 Chosen Plaintexts", Journal of Cryptology, Volume 2, Number 3, 1990.

4. Position Location and Reporting System - System Technical Description, U.S Army Communications Research and Development Command, Fort Monmouth, New Jersey, Document FR 81-14-251, March 1981.

5. Peterson and Brown, "Cyclic Codes for Error Detection", Proceeding of the IRE, January 1961, pp. 228-235.

6. Meyer and Matyas, Cryptography: A New Dimension in Computer Security, Wiley Interscience, New York, 1982, p. 61.

7. Golomb, Shift Register Sequences, Aegean Park Press, Laguna Hills, CA, 1982.

8. Scholtz and Welch, "Continued Fractions and Berlekamp's Algorithm", IEEE Trans. Inform. Theory, IT-15, pp. 90-94, January 1974.

9. Key, "An Analysis of the structure and complexity of nonlinear binary sequence generators", IEEE Trans. Inform. Theory, IT-22, pp. 732-736, November 1976.

10. Peterson and Weldon, Error Correcting Codes, MIT Press, Cambridge, MA, 1972.

11. Fredricksen, "Pseudo randomness properties of binary shift register sequences", IEEE Trans. Inform. Theory, pp. 115-120, January 1975.

12. Beker and Piper, Cipher Systems, Wiley Interscience, New York, 1982, Chapter 4.

13. Method and Apparatus for Securing Integrated Circuits from Unauthorized Copying and Use, Hughes Aircraft Company, 23 August 1988, Patent Number 4,766,516.

KNOWLEDGE-BASED INFERENCE CONTROL IN A MULTILEVEL SECURE DATABASE MANAGEMENT SYSTEM

Bhavani Thuraisingham

The MITRE Corporation, Burlington Road, Bedford, MA

1. INTRODUCTION

Inference problem is the problem of users deducing unauthorized information from the legitimate information that they acquire. We are particularly interested in the inference problem which occurs in a multilevel operating environment. In such an environment, users are cleared at different security levels and they access a multilevel database where the data is classified at different sensitivity levels. A multilevel secure database management system (MLS/DBMS) manages a multilevel database where its users cannot access data to which they are not authorized. However, providing a solution to the inference problem, where users issue multiple requests and consequently infer unauthorized knowledge, is beyond the capability of currently available MLS/DBMSs.

Due to the complexity of the inference problem (see for example [THUR90a]), we believe that a triple approach to research is needed to combat it; one is to build inference controllers which act during transaction processing, the other is to build inference controllers for database design, and the third is to build inference controllers to act as advisors to the System Security Officer (SSO). In our recent papers, we have described prototypes for handling the inference problem during query and update processing [FORD90, COLL90]. In addition, techniques for handling this problem during database design have also been proposed [THUR91a]. While the previous approaches enable the detection and/or prevention of simple inference strategies that users could utilize to draw inferences, we believe that for an inference controller to be effective, it should be able to capture the complex reasoning strategies of humans. In other words, what is needed is a knowledge-based inference controller.

Knowledge-based inference control is a two-step process. The first step is to represent the multilevel application as completely and accurately as possible. The second step is to reason about the application so that security violations via inference could be prevented and/or detected. In section 2 of this paper we discuss the use of conceptual graphs for representing the multilevel application. A tool based on conceptual graphs could be utilized by the SSO to design the multilevel database application. While the computation techniques developed for conceptual graphs could be utilized for reasoning about the multilevel database application, the output from the MLS/DBMS also plays a significant role in users making unauthorized deductions. This means that any reasoning tool must also take into consideration the responses released by the MLS/DBMS and audit data in order to effectively prevent/detect security violations via inference. In section 3 of this paper we discuss the essential points towards designing such a tool. Figure 1 illustrates the two step process involved in knowledge-based inference control. We envisage that a tool based on the approach described here could be utilized by the SSO to detect/prevent security violations via inference. The front-end of the tool represents the multilevel database application, responses released by the MLS/DBMS, and the audit data in a format that can be understood by the SSO. The back-end of the tool reasons with the knowledge and detects/prevents certain security violations via inference.

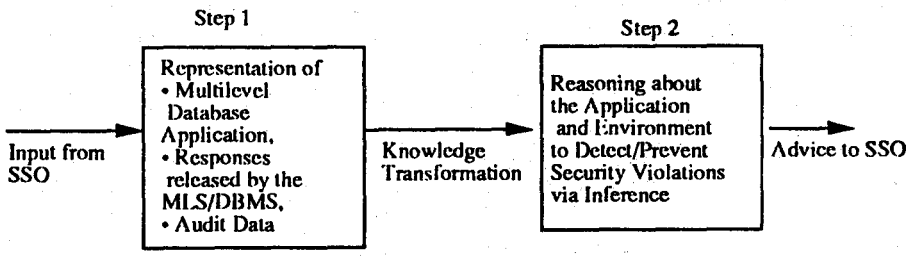


Figure 1. Knowledge-based Inference Control

2. REPRESENTING AND REASONING ABOUT MULTILEVEL DATABASE APPLICATIONS

2.1 BACKGROUND

We have utilized conceptual structures for representing and reasoning about multilevel database applications. In particular, we have examined the use of semantic nets as well as conceptual graphs for this purpose. The use of conceptual structures for inference control was first proposed by Hinke [HINK88] where the use of graph theoretic techniques was described. Later Smith [SMIT90] investigated the use of semantic data models for representing multilevel applications. The work reported in [BUCZ89] also investigated the use of semantic data modeling techniques for controlling inferences in a multilevel environment. The use of conceptual graphs to handle the inference problem was first introduced in [THUR90b] and later in [HINK92]. Other work on the use of conceptual structures for representing and/or reasoning about multilevel database applications is reported in [BINN92, GARV92, SELL92].

Among the various conceptual structures such as semantic nets, semantic data models, and conceptual graphs, conceptual graphs seem to be the most appropriate scheme for representing complex applications. This is because conceptual graphs subsume other structures such as semantic nets and they have the full power of first order logic. Unlike logic-based systems, conceptual graphs represent knowledge in a manner similar to the way humans view the world. Furthermore, they can also be extended to include modality and time without much difficulty. Another advantage of using such a scheme is that the techniques developed for reasoning with conceptual graphs could be utilized for detecting security violations via inference (see for example the discussion in [SOWA84]).

We have chosen conceptual graphs for representing multilevel database applications. Although reasoning with conceptual graphs is as powerful as reasoning with a logic programming system, most of the current knowledge-based systems are not based on conceptual graphs. Therefore, in our approach, the back-end of the inference controller, shown in figure 1, reasons with knowledge represented in the form of rules and frames. In other words, the conceptual graph representation utilized by the front-end of the inference controller must be transformed into frames and rules in order to be processed by the back-end. The use of conceptual graphs is described in section 2.2. The back-end of the inference controller is described in section 3.

2.2 THE USE OF CONCEPTUAL GRAPHS

The use of conceptual graphs for handling the inference problem was first proposed in [THUR90b]. However, in [THUR90b], the use of inference rules for conceptual graphs to detect security violations via inference was not addressed. In this section, we review some of the essential points in conceptual graphs for representing multilevel database applications, and discuss with an example how security violations may be detected.

As stated in [SOWA84], a conceptual graph is a finite connected bipartite graph which consists of concepts and conceptual relations. Every conceptual relation has one or more arcs, each of which is linked to a concept. We define a multilevel conceptual graph to be a conceptual graph in which some of the concepts and conceptual relations are sensitive. Figure 2 shows a multilevel conceptual graph (which was represented using a semantic net in [THUR90b]). The Unclassified interpretation of this graph is as follows: CHAMPION carries passengers. Its captain is Smith who has 20 years' experience. The ship is located in the Mediterranean Sea on 16 June 1990. It's destination is Greece. The Secret interpretation is as follows: CHAMPION carries SPARK which is an explosive. Its captain is Smith who has battle management experience. The ship is located in the Mediterranean Sea on 16 June 1990. Its destination is Libya. (Note that the Secret concepts and relations are illustrated by darkened structures and lines.)

In [THUR90b], some formation rules (for example, the join of two conceptual graphs, adding connectives such as negation to a conceptual graph) were discussed. These formation rules produce new conceptual graphs. However, these formation rules do not enable any computation. In order to detect security violation via inference, some form of computation with conceptual graphs needs to be performed. In [SOWA84], several types of rules of inference have been proposed for conceptual graphs. These rules enable computation with conceptual graphs. Figure 3 illustrates a deduction rule similar to Modus Ponens in logic. Figure 3(a) illustrates at the Unclassified level the fact that if CHAMPION is sailing to Libya, then it must be a warship. Figure 3(b) illustrates at the Secret level the fact CHAMPION is a warship and at the Unclassified level the fact that it is a passenger ship. Figure 3c illustrates at the Unclassified level the fact that Champion is sailing to Libya. The set of graphs shown in figure 3 is inconsistent as

there is contradictory information at the Unclassified level. If a set of graphs is inconsistent, then there is a potential for a security violation via inference.

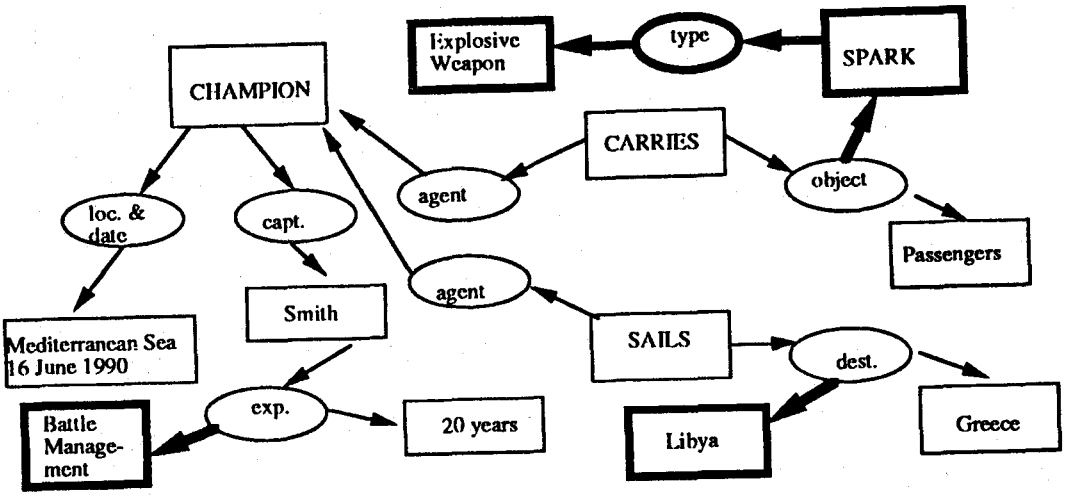


Figure 2. Multilevel Conceptual Graph

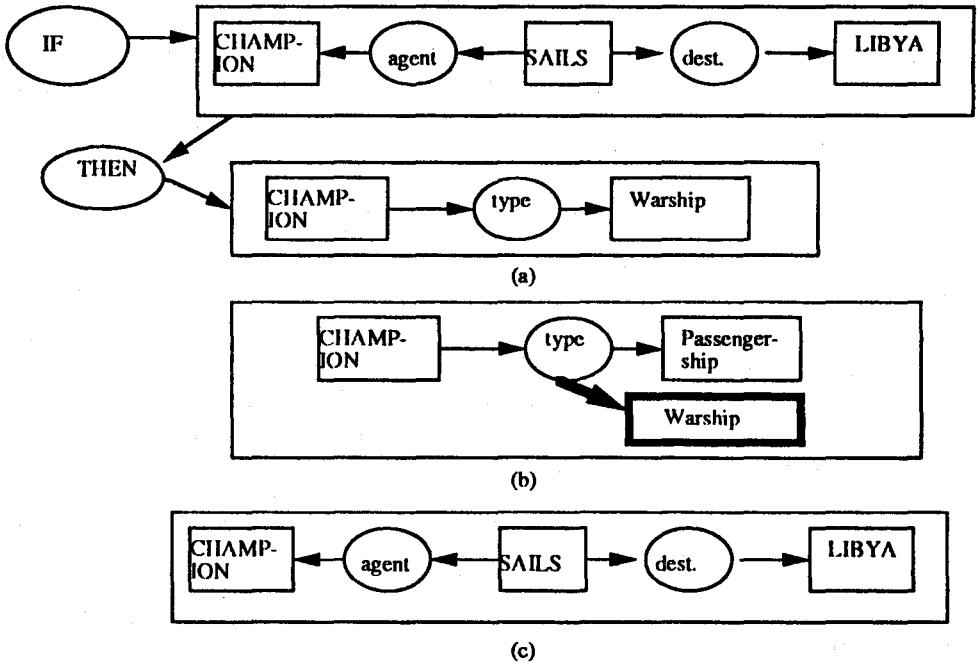


Figure 3. Inconsistent Set of Conceptual Graphs

3. KNOWLEDGE-BASED INFERENCE CONTROL

In this section, we discuss the issues involved in designing the back-end of the inference controller illustrated in figure 1. We will call this module the knowledge-based inference controller (KBIC). In section 3.1, we describe the modules of the system. Knowledge representation issues are discussed in section 3.2. Reasoning techniques are described in section 3.3. Issues on truth maintenance are addressed in section 3.4. Implementation issues are discussed in section 3.5. Much of our work has been influenced by the Cyc project carried out at MCC [LENA89]. A discussion on knowledge-based inference control is also given in [THUR91b].

3.1 MODULES

The major modules of the KBIC are shown in figure 4. They are: the User Interface (UI), the Knowledge Manager (KM), the Inference Engine (IE), the Conflict/Contention Resolution System (CCRS), and the Truth Maintenance System (TMS), A description of each module is given below.

UI is the interface to the KBIC. It can be used for updating the knowledge base, for querying, for obtaining advice from the KBIC, or for requesting the KBIC to solve a particular problem. UI is also used if additional information is required from the SSO. Furthermore, UI is the module which interfaces to the tool which is used to represent the multilevel database application discussed in section 2. KM is responsible for managing and structuring the knowledge base. It must also ensure the consistency of the knowledge base. Any access to the knowledge base is via KM. It has interfaces to all of the modules of the KBIC. The knowledge base stores all of the relevant information. This includes security constraints, real-world information, heuristics, and relevant information released to various users. IE is the heart of the KBIC. It has the potential for using a variety of inference strategies. As a minimum, IE should be able to perform logical inferences. Note that in a multilevel environment, there could be different views of the same entity at different security levels. This means that the knowledge base could potentially have conflicting information about an entity at conceptually different security levels. Therefore IE should be able to reason across security levels. CCRS is responsible for resolving conflicts as well as determining the best choice to take when the system is presented with different options. For example, one particular reasoning strategy could potentially give results which conflict with another reasoning strategy. In such a situation, IE would consult CCRS to resolve the conflict. The conflict is resolved by CCRS querying either the KM or even the SSO if necessary. TMS is the module that is responsible for maintaining the consistency of the various beliefs. Such a module is necessary for nonmonotonic reasoning.

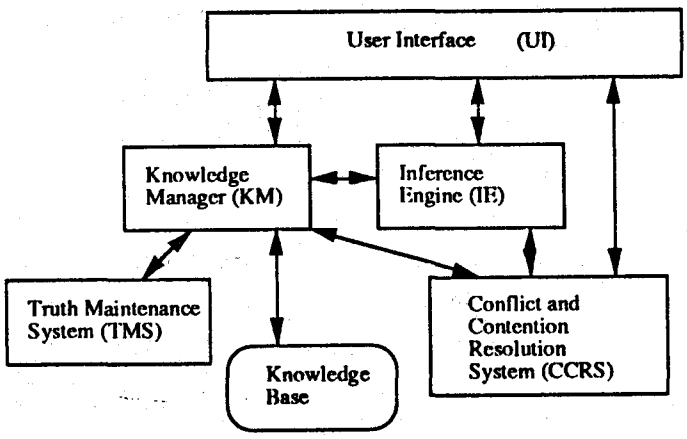


Figure 4. Modules of the KBIC

3.3 KNOWLEDGE REPRESENTATION

The knowledge representation scheme used by the KBIC is a combination of frames and rules. Frames are ideal to represent structured knowledge. The inheritance mechanism in frames is a powerful one which enables the representation of generic entities, as well as instantiations of the generic entities. The frames used to represent the knowledge are called knowledge frames. Each knowledge frame describes a generic entity or a specific instance of a generic entity. A knowledge frame has many slots associated with it. Each slot describes some property of the entity represented or it could have rules or security constraints associated with it.

Every knowledge frame has one slot which specifies the security level at which the knowledge frame is true. Furthermore, since users at different levels could have different views of the same entity, frames at different levels are used to represent such views. Figure 5 shows Unclassified and Secret knowledge frames which have information on the ship CHAMPION. Since CHAMPION is a ship, it inherits information from the knowledge frame which has information on the generic entity SHIP. Each knowledge frame also has real-world information and security constraints associated with it. Note that whenever the word "inherit" is used for a slot, it means that the value for that slot is inherited from the knowledge frame representing the generic entity of the specific instance.

Name of Entity: SHIP;
Entity Type: Generic
Security Level: Unclassified
Information in Database: Ship#, Ship-name, Mission#
Other Information: None
Security Constraints: None
Instances: CHAMPION, - - - -

Name of Entity: CHAMPION;
Entity Type: Instance of SHIP
Security Level: Unclassified
Information in Database: Inherit
Other Information:
(i) The destination is Greece
(ii) If destination is Libya there will be war
(iii) If ship is in the Pacific, then it cannot go to Libya.
(iv) Inherit
Security Constraints: If destination is Libya then all mission related information of CHAMPION is Secret

Name of Entity: CHAMPION;
Entity Type: Instance of SHIP
Security Level: Secret
Information in Database: Inherit
Other Information:
(i) The destination is Libya
(ii) If destination is Libya there will be war
(iii) If ship is in the Pacific, then it cannot go to Libya.
(iv) Inherit
Security Constraints: Inherit

Figure 5. Knowledge Frames

Name of Entity: CHAMPION
Entity Type: SHIP
Security Level: Unclassified
Location: Mediterranean Sea
Date: June 16, 1990
Destination: Greece
Carries: Passengers
Captain: Smith

Name of Entity: Smith
Entity Type: Captain
Security Level: Unclassified
Skills: 20 Years Experience

Name of Entity: Smith
Entity Type: Captain
Security Level: Secret
Skills: Battle Management

Name of Entity: CHAMPION
Entity Type: SHIP
Security Level: Secret
Location: Mediterranean Sea
Date: June 16, 1990
Destination: Libya
Carries: Spark
Captain: Smith

Name of Entity: Spark
Entity Type: Weapon
Security Level: Secret
Weapon Type: Explosive

Figure 6. Transformed Knowledge Frames

In addition to representing knowledge as frames, rules are also used to represent some of the knowledge such as constraints, real-world data, and conflict resolution. The rules could be specified in a logic-based language such as datalog [ULLM88].

Representing the multilevel database applications as well as the input from the MLS/DBMS in the form of frames and rules may not be straightforward for complex applications. Therefore, representing the application first using conceptual structures such a conceptual graphs will ease the burden placed o the knowledge engineer. The tool which represents the multilevel database application described in section 2 bridges the semantic gap between the world and the knowledge base. Furthermore, tools have been developed to transform applications represented using conceptual structures into frames and rules [SOWA84]. In figure 6, we show how the graph of figure 2 may be represented as a collection of frames.

3.3 REASONING

The KBIC uses rule-based reasoning and frame-based reasoning. In addition, it also reasons across security levels. Some of the essential points are discussed in this section. In order for the inference controller to be effective it must also reasoning under uncertainty and utilize additional inference strategies such as inductive and heuristic reasoning. Such reasoning techniques will be part of the future investigation.

Rule-based reasoning techniques include forwards chaining, backward chaining, and hybrid approaches. We illustrate how security violations via inference may be detected with a simple example. Consider an Unclassified rule base consisting of the following two rules:
R1: CHAMPION is a warship
R2: If X is a warship, its mission is Secret
R3: CHAMPION's mission is Iraq Crisis
Suppose an unclassified user is given the information R1 and R2. Then this release of information must also be recorded in the knowledge base (by KM). IE could reason as follows: since CHAMPION is a warship, using rule R2, its mission is Secret. Since CHAMPION's mission is Iraq crisis, this mission must be kept Secret. Since CHAMPION's mission has been given to an Unclassified user, a security violation has occurred.

As stated in [FROS86], the problem solving technique used by frame-based systems is "matching." Given some information about an entity in the real world, the system will try to match the values associated with the entity with the slot values of frames. We illustrate how security violations via frame-based inference could occur with a simple example. Consider an Unclassified frame which describes all of the properties of a passenger ship named CHAMPION. Suppose OHIO is another ship and there is a security constraint that classifies all properties of OHIO at the Secret level. There is also an Unclassified rule which states that OHIO and CHAMPION are similar. From this rule, an Unclassified user could infer some of the Secret properties of OHIO. Therefore, one should classify the fact that OHIO and CHAMPION are similar at least at the Secret level.

As stated in section 3.1, IE should be able to reason across security levels. In the example of figure 5, when IE is reasoning at the Unclassified level (i.e. to detect/prevent unauthorized inferences that users at the Unclassified level could make) it considers the knowledge frame on CHAMPION at Unclassified level. If it is reasoning at the Confidential level, then it still considers the knowledge frame at the Unclassified level, as there is no knowledge frame on CHAMPION at the Confidential level. If it is reasoning at the Secret level, then it could do one of the following:
- Consider only the knowledge frame on CHAMPION at the Secret level.
- Consider both the knowledge frames on CHAMPION at the Unclassified and Secret levels.
- Consult with CCRS as to which frame to consider.

A simple solution would be to take the first action. That is, assume that information at level L is more accurate than the information at level L-1. In reality, however, information at a lower level could be more accurate. For example, information at a lower level could be more current than the one at the higher level. CCRS could resolve the conflicts either by (i) checking the knowledge base for appropriate conflict resolution rule, (ii) querying the user to give more up-to-date information, (iii) in the absence of appropriate information, make heuristic guesses based on recent experiences, and (iv) reason using the rules of a theory such as plausibility theory [FROS86].

3.4 ISSUES ON TRUTH MAINTENANCE

TMS is the module of the KBIC that is responsible for maintaining the consistency of the various beliefs. Such a module is necessary for nonmonotonic reasoning. In this section we discuss the essential points in extending Doyle's Truth Maintenance System (TMS) [DOYL82] to reason in a multilevel environment.

In TMS, statements of belief are called 'nodes.' Each node (or statement of belief) is assigned a security level. If a node is assigned a security level L, then it can be IN or OUT with respect to any level ≥ L. A node is IN with respect to L if it is believed to be true at L. Otherwise, the node is OUT. Each node at level L has a set of justifications linked to it with respect to each security level that dominates L. Each justification at a level L* ≥ L represents a justification representing one way in which the node (i.e., the belief which corresponds to it) may be true. If a justification at level L* is valid, then, unless that justification is explicitly made invalid at level L** (L** is the least level which dominates L*), it is also assumed valid at level L**. A node at level L is IN with respect to level L*≥ L if it has at least one justification valid at L*. If all justifications at level L* are not valid, then the node is OUT with respect to L*.

We illustrate the essential points of a truth maintenance with an example. In this example, we assume that there are only two security levels, Unclassified (U) and Secret (S). Figure 7 shows the TMS nodes and justifications at the Unclassified level. This figure is interpreted as follows. The nodes are numbers 1 through 4. Each node has the following assertion or belief. Node 1 has the assertion "CHAMPION is a ship." This assertion has the status IN and does not have any justifications associated with it. Node 2 has the belief "CHAMPION sails to Japan." In order for this belief to be IN, node 1 must be IN and node 3 must be OUT. Node 1 is IN. We will see that node 3 is OUT. Therefore, Node 2 is IN. That is, CHAMPION sails to Japan is consistent with everything that is believed with respect to the Unclassified level. Node 3 has the belief "CHAMPION is not a passenger ship." In order for this belief to be true, node 4 must be IN. We will see that node 4 is OUT. Therefore, Node 3 is OUT. Node 4 is a previous assertion "CHAMPION carries explosives." It has the status OUT because it must have been retracted earlier.

	Justification		
Node	Status	IN	OUT
1. Champion is a ship	IN		
2. Champion sails to Japan	IN	1	3
3. Champion is not a passenger ship	OUT	4	
4. Champion carries explosives	OUT		

Figure 7. Justifications at the Unclassified Level

		Justification 1		Justification 2	
Node	Status	IN	OUT	IN	OUT
1. Champion is a ship	IN				
2. Champion sails to Japan	OUT	1	3		
3. Champion is not a passenger ship	IN	4		5	
4. Champion carries explosives	OUT				
5. Champion is a warship	IN				

Figure 8. Justifications at the Secret Level

Figure 8 shows the assertions, beliefs, and justifications at the Secret level. Here there are two justifications that could possibly be associated with a node. This table is interpreted as follows. There are four unclassified nodes (i.e., beliefs) as in the Unclassified world and one Secret node. Node 1 has the assertion "CHAMPION is a ship." This assertion has the status IN and does not have any justifications associated with it. Note that node 1 is assigned the Unclassified level. Its status has not changed from the Unclassified world. Node 2 has the belief "CHAMPION sails to Japan." In order for this belief to be IN, node 1 must be IN and node 3 must be OUT. Node 1 is IN. We will see that node 3 is also IN. Therefore, Node 2 is OUT. That is, CHAMPION sails to Japan is not consistent with everything that is believed with respect to the Secret level. Note that node 2 is assigned the Unclassified level. Its status has changed from the Unclassified world. Node 3 has the belief "CHAMPION is not a passenger ship." In order for this belief to be true, either node 4 must be IN or node 5 must be IN. We will see that node 5 is IN. Therefore, Node 3 is IN. That is, "CHAMPION is not a passenger ship" is consistent with everything that is believed with respect to the Secret level. Note that node 3 is assigned the Unclassified level. Its status has changed from the Unclassified world. Node 4 is a previous assertion "CHAMPION carries explosives." It has the status OUT because it must have been retracted earlier. Note that node 4 is assigned the Unclassified level. Its status has not changed from the Unclassified world. Node 5 is an assertion "CHAMPION is a warship." It has the status IN. Note that node 5 is assigned the Secret level and is, therefore, not visible at the Unclassified level.

If at a later time the assertion that "CHAMPION is a warship" is retracted in the Secret world, then the status of node 5 becomes OUT. This would change the status of node 3 to be OUT. This would, in turn, change the status of node 2 to be IN. It should also be noted that a TMS does not create justifications. The justifications are provided by KM to TMS. The TMS maintains a consistent set of beliefs with respect to all security levels.

3.5 IMPLEMENTATION ISSUES

One of the ways to implement the KBIC would be to use an existing expert system shell. Commercial off-the-shelf expert system shells such as G2 (product of Gensym Inc.) are now available. Many of these shells handle knowledge bases represented as frames and rules. While using a commercial shell has obvious advantages, such as reduced implementation time and effort, it may not be tailored to solve special problems. That is, one has to contend with the reasoning strategies implemented by the inference engine of the shell. Any additions and/or enhancements to the reasoning strategies may be quite complex to implement. Also, one would need the source code of the shell to make these enhancements. Therefore, unless we can find a shell that can specifically handle the reasoning strategies of the KBIC, this may not be a desired approach. Another approach is to implement the KBIC in a conventional language such as C. While implementation in C has obvious advantages with respect to efficiency, some of the complex reasoning strategies and data structures may be difficult to implement.

A third approach is to use an AI language such as Lisp or Prolog. While both languages have their advantages and disadvantages, since we are mainly interested in handling the inference problem in a relational database management system, the preferred language seems to be Prolog. This is because there is a natural relationship between the Prolog data model and the relational data model. In fact, a relational database is a Prolog program [LLOY87]. Prolog interfaces to relational database systems are increasing [LI84, ICOT87]. Furthermore, all of the essential features of the KBIC, such as reasoning under uncertainty, truth maintenance, and handling frame and rule-based representations, can be implemented in Prolog (see for example the discussion in [MERR89]). For these reasons, Prolog may be an appropriate language to implement the KBIC.

4. SUMMARY AND FUTURE CONSIDERATIONS

In this paper, we have described the inference problem in multilevel database management systems, identified the needs for knowledge-based inference control, and discussed the issues involved in developing a knowledge-based inference controller. Building a knowledge-based inference controller is a two-step process. The first step is to represent the multilevel database application. The second step is to develop techniques for reasoning about the application. We first proposed the use of conceptual structures, such as conceptual graphs, for representing the application. Such a scheme was proposed as it was a natural way to model the world and it had the full power of first order logic. Then we described the essential points of the module which reasons with the knowledge represented in the from of frames and rules. In order for the inference controller to function effectively, the knowledge represented as a collection of conceptual graphs must be transformed into frames and rules.

The developments in artificial intelligence techniques show much promise for the design and development of inference controllers. There is still much work to be done on knowledge representation, knowledge transformation, reasoning under uncertain and incomplete information, and handling different types of inference strategies that users could utilize to draw unauthorized inferences.

ACKNOWLEDGMENT

We thank William Ford for useful discussions on knowledge-based inference control. We also thank Rae Burns, Penny Chase, Deborah Conte, and William Ford for their comments on this paper. We gratefully acknowledge the Department of the Navy (SPAWAR) for sponsoring our initial investigation on the use of knowledge-based techniques for handling the inference problem under contract F19628-89-C-0001.

REFERENCES

[BINN92] Binns, L., August 1992, Inference Through Secondary Path Analysis," Proceedings of the 6th IFIP Working Conference in Database Security, Vancouver, British Columbia.

[BUCZ89] Buczkowski, L. J., and E. L. Perry, February 1989, *Database Inference Controller*, Interim Technical Report, Ford Aerospace Corporation.

[COLL90] Collins, M., October 1990, *Design and Implementation of a Secure Update Processor*, Technical Report MTR10977, The MITRE Corporation (a version published in the Proceedings of the 7th Computer Security Applications Conference - coauthors: W. Ford and B. Thuraisingham).

[DOYL82] Doyle, J., 1982, "A Truth Maintenance System," *Artificial Intelligence Journal*, Vol. 12.

[FORD90] Ford, W. R., J. O'Keeffe, and B. Thuraisingham, August 1990, *Database Inference Controller: An Overview*, Technical Report MTR10963 Vol. 1, The MITRE Corporation.

[FROS86] Frost R., 1986, *Introduction to Knowledge-Base Management Systems*, Collins, London.

[GARV92] Garvey, T., et al, August 1992, Toward a Tool to Detect and Eliminate Inference Problems," Proceedings of the 6th IFIP Working Conference in Database Security, Vancouver, British Columbia.

[HINK88] Hinke, T., April 1988, "Inference Aggregation Detection in Database Management Systems," Proceedings of the IEEE Symposium on Security and Privacy.

[HINK92] Hinke T., and H. Delugach, August 1992, "Aerie: An Inference Modeling and Detection Approach for Databases," Proceedings of the 6th IFIP Working Conference in Database Security, Vancouver, British Columbia.

[ICOT87] [ICOT87] "ICOT Project," 1987, *New Generation Computing Journal*, Vol. 5.

[LENA89] Lenat, D., and R. Guha, 1989, "*Building Large Knowledge-Based Systems*," Addison Wesley, MA.

[LI84] Li, D., 1984, *A Prolog Database System*, Research Studies Press, John Wiley and Sons, London.

[LLOY87] Lloyd, J., 1987, *Foundations of Logic Programming*, Springer Verlag, Heidelberg, Germany.

[MERR89] Merritt, D., 1989, *Building Expert Systems In Prolog*, Springer Verlag, New York.

[SELL92] Sell, P., August 1992, "The Spear Data Design Method," Proceedings of the 6th IFIP Working Conference in Database Security, Vancouver, British Columbia.

[SMIT90] Smith, G., May 1990, "Modelling Security-Relevant Data Semantics," Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland, CA.

[SOWA84] Sowa, J., 1984, *Conceptual Structures: Information Processing in Minds and Machines*, Reading, MA: Addison Wesley.

[THUR90a] Thuraisingham, B., June 1990, "Recursion Theoretic Properties of the Inference Problem in Database Security," Presented at the Third IEEE Computer Security Foundations Workshop, Franconia, NH (also available as MITRE Technical Paper MTP-291).

[THUR90b] Thuraisingham, B., August 1990, "The Use of Conceptual Structures to Handle the Inference Problem," M90-55, The MITRE Corporation (a version also published in the Proceedings of the 5th IFIP Working Conference in Database Security).

[THUR91a] Thuraisingham, B., April 1991, "Handling Security Constraints during Multilevel Database Design," Proceedings of the 4th RADC Database Security Workshop, Little Compton, RI.

[THUR91b] Thuraisingham, B., and W. Ford, October 1991, "Issues on the Design and Implementation of an Intelligent Database Inference Controller," Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Charlottesville, VA.

[ULLM88] Ullman, J., 1988, *Principles of Database and Knowledge-Base Systems*, Computer Science Press, Rockville, MD.

A LATTICE INTERPRETATION OF THE CHINESE WALL POLICY

Ravi S. Sandhu[1]

Center for Secure Information Systems
&
Department of Information and Software Systems Engineering
George Mason University, Fairfax, Virginia 22030

Abstract. The typed access matrix (TAM) model was recently defined by Sandhu. TAM combines the strong safety properties for propagation of access rights obtained in Sandhu's Schematic Protection Model, with the natural expressive power of Harrison, Ruzzo, and Ullman's model. In this paper we consider the implementation of TAM in a distributed environment. To this end we propose a simplified version of TAM called Single-Object TAM (SO-TAM). We illustrate the practical expressive power of SO-TAM by showing how the ORCON policy for originator control of documents can be specified in SO-TAM. We provide arguments to support our conjecture that SO-TAM is theoretically as expressive as TAM. We show that SO-TAM has a simple implementation in a typical client-server architecture. Our design is based on access control lists as the principal means for enforcing access to subjects and objects. In addition, certificate servers are introduced for generating certificates for checking access rights in those cases where access control lists are insufficient. A major advantage of our design is that atomicity of operations does not require a distributed commit.

Keywords: Access Matrix, Distributed Systems, Secure Architectures, ACLs, Certificates

1 INTRODUCTION

Distributed systems have become the prevalent mode of computing. Modern systems offer a great deal of flexibility in tailoring a user's environment. The physical distribution of data and other resources can be made as transparent as a user wishes. It is important that security researchers and practitioners provide similar flexibility with respect to access control mechanisms.

To provide flexibility in access control we first need a flexible model which can express a rich variety of security policies. In our opinion flexibility is achieved by allowing users to propagate access rights to other users, with a combination of discretionary and mandatory controls. We would like to give individual users as much discretionary choice as possible, within the constraints required to meet the overall objectives and policies of an organization. For example, members of a project team might be allowed to freely share project documents with each other, but only the project leader is authorized to allow non-members to read project documents.

Security models based on propagation of access rights must confront the safety problem. In its most basic form, the safety question for access control asks: is there a reachable state in which a particular subject possesses a particular right for a specific object? There is an essential conflict between the expressive power of an access control model and tractability of safety analysis. The access matrix model as formalized by Harrison, Ruzzo, and Ullman (HRU) [5] has very broad expressive power. Unfortunately, HRU also has extremely weak safety properties.

Recently Sandhu [9] has shown how to overcome the negative safety results of HRU by introducing

[1]This work was partially supported by the National Security Agency through contract MDA904-92-C-5141.

Largely due to this dynamic aspect, Brewer and Nash claim that the Chinese Wall policy "cannot be correctly represented by a Bell-LaPadula model." One objective of our paper is to dispute this claim, by showing how the Chinese Wall policy is just another example of a lattice-based information flow policy which can be easily represented within the Bell-LaPadula framework.[2]

Another objective of our paper is to show the vital importance of distinguishing security policy as applied to human users versus security policy as applied to computer subjects. Brewer and Nash fail to make this distinction. They treat users and subjects as synonymous concepts. As a result their model is much too restrictive to be employed in a practical system. By maintaining a careful distinction between users, principals and subjects, we develop a model for the Chinese Wall policy which addresses threats from Trojan Horse infected programs. The Brewer-Nash model on the other hand makes a futile attempt to safeguard against malicious consultants.

The rest of this paper is organized as follows. Section 2 reviews the distinction between users, principals and subjects in a computer system. Section 3 discusses the Chinese Wall policy and the threats that it addresses. We carefully distinguish between threats posed by malicious consultants versus threats posed by Trojan Horse infected programs. While computer security can address threats posed by Trojan Horse infected programs, it cannot fully address threats posed by malicious consultants. After all, consultants who choose to share information in violation of Chinese Walls can do so equally efficiently by communication means outside of the computer system. With this context we analyze the Brewer-Nash model in section 4 and show that this model is unduly restrictive. In section 5 we develop a lattice-based model for the Chinese Wall policy and relate it to the Bell-LaPadula model. Section 6 concludes the paper.

2 USERS, PRINCIPALS AND SUBJECTS

To understand the Chinese Wall policy and its nuances with respect to subjects versus human users, we must first understand the distinction between *users*, *principals* and *subjects*. This distinction is fundamental to computer security and goes back to the beginnings of the discipline. Nevertheless, it is often dealt with imprecisely in the literature leading to undue confusion about the objectives of computer security.

2.1 Users

We understand a user to be a human being. We assume that each human being known to the system is recognized as a unique user. In other words the unique human being Jane Doe cannot have more than one user identity in the system. If Jane Doe is not an authorized user of the system she has no user identity. Conversely, if she is an authorized user she is known by exactly one user identity, say, JDoe. Clearly this assumption can be enforced only by adequate administrative controls, which we assume are in place. It should be noted that violation of this requirement is often the cause of security violations in current systems.

2.2 Principals

Our concept of principal is adapted from Saltzer and Schroeder [6]. Each user may have several principals associated with the user. On the other hand each principal is required to be associated with a single user.

[2] In fairness to Brewer and Nash it should be noted that the original Bell-LaPadula model is inadequate to express the Chinese Wall policy. The model given here does require (i) a careful distinction between users, principals and subjects, and (ii) the concept of user labels which "float up" versus labels on principals, subjects and objects which do not change.

The motivation in [6] for this concept was that different principals would correspond to, say, different projects on which the user works. Every time a user logs in to the system it is as a particular principal. Thus if Jane Doe was assigned to projects Red and Blue, she would have three principals associated with her user identity, say, JDoe, JDoe.Red and JDoe.Blue. On any session Jane could login as any one of these principals, depending on the work she planned to do in that session. Each principal associated with JDoe obtains a different set of access rights. Thus JDoe.Red has access to the files and other objects of project Red, but not project Blue. Similarly, JDoe.Blue has access to the files and other objects of project Blue, but not project Red. The principal JDoe is a generic principal for Jane allowing access to her personal files, but not to any of the project files.

The notion of principal reflects the everyday reality that individuals wear several different "hats" in an organization, with their authority and responsibility determined by the particular "hat" they are wearing at a given moment. Saltzer and Schroeder introduce principals in a discretionary context. The concept carries over equally well to mandatory policies. We often encounter phrases such as, "the top-secret user John logs in at the secret level," in the security literature. What are we to make of this statement? In the user-principal terminology we interpret this statement as follows:

- Firstly, there is a unique user John, cleared to top-secret, independent of the level at which John logs in.
- Secondly, John can log in at every level dominated by top-secret. At each of these levels there is a separate principal associated with John. So John.top-secret is the principal when John logs in at top-secret, John.secret is the principal when John logs in at secret, etc.

We will see that this concept of a principal is the key to achieving lattice-based enforcement of Chinese Walls.

2.3 Subjects

We understand a subject to be a process in the system, i.e., a subject is a program in execution. Each subject is associated with a single principal on behalf of whom the subject executes. In general a principal may have many subjects associated with it concurrently running in the system.

For simplicity we assume that a subject executes with all the privileges of its associated principal.[3] Thus when Jane Doe logs in as JDoe.Red and invokes her favorite editor Emacs, a subject associated with JDoe.Red is created and runs the Emacs code. This subject acquires all the access rights of the principal JDoe.Red. Similarly when John logs in as John.top-secret every subject spawned during that session runs at the top-secret level.

To summarize

- each authorized human user is known as a unique user to the system,[4]
- each user can log in as one of several principals but each principal is associated with only one user, and
- each principal can spawn several subjects but each subject is associated with only one principal.

[3] This is the actual situation in most existing systems, including those specifically designed for security. More generally a subject could be created with a proper subset of privileges of its associated principal. The most general case is to allow a subject to have multiple parents, from each of whom it obtains some privileges.

[4] This requirement is admittedly violated in many systems, and will require administrative controls outside of the computer system. Nevertheless, without this requirement there is little scope for enforcing aggregation policies such as Chinese Walls. Moreover, it is also a prerequisite for enforcing separation of duties.

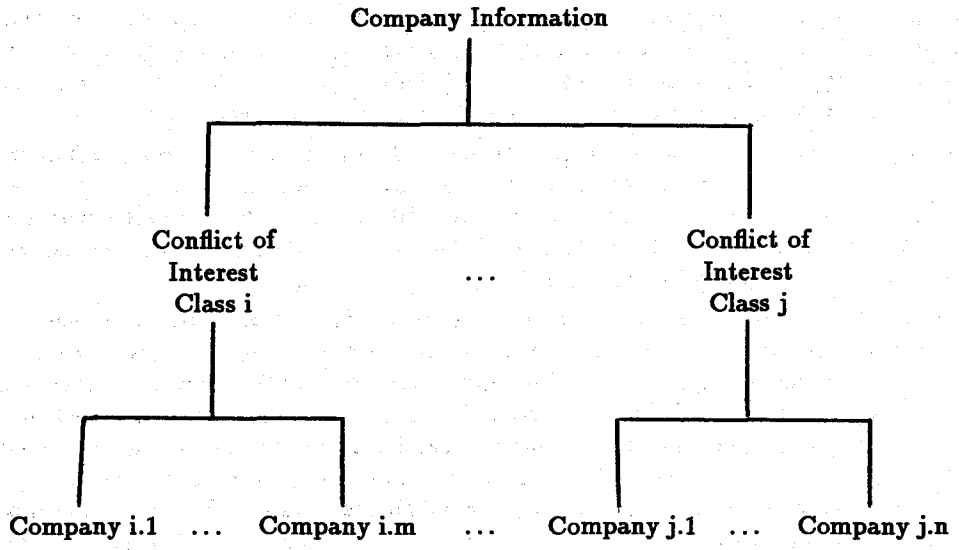


Figure 1: Company Information in the Chinese Wall Policy

3 THE CHINESE WALL POLICY

The Chinese Wall policy is intuitively simple and easy to describe. In this section we describe this policy by adapting the description of Brewer and Nash [2] and adding additional concepts to it. It is important to keep in mind that we are deliberately ignoring all discretionary access control issues in this paper. In practice the Chinese Wall policy as described here would be the mandatory component of a larger policy which includes additional discretionary controls (and possibly additional mandatory controls).

We begin by distinguishing *public* information from *company* information. There are no mandatory controls on reading public information. Reading company information on the other hand is subjected to mandatory controls, which we will discuss in a moment. The policy for writing public or company information is derived from its consequence on providing possible indirect read access contrary to the mandatory read controls. It is in this respect that users and subjects must be treated differently. We will consider mandatory controls on writing information following our discussion of the read controls.

The motivation for recognizing public information is that a computer system used for consulting services will inevitably have large public databases for use by consultants. Moreover, public information allows for desirable features such as public bulletin boards and electronic mail which users expect to be available in any modern computer system. Public information can be read by all users, principals and subjects in the system (restricted only by discretionary controls which, as we have said, we are ignoring in this paper).

Company information is categorized into mutually disjoint conflict of interest classes as shown in figure 1. Each company belongs to exactly one conflict of interest (COI) class. The Chinese Wall policy requires that a consultant should not be able to read information for more than one company in any given COI class. To be concrete let us say that COI class i consists of banks and COI class j consists of oil companies. The Chinese Wall stipulation is that the same consultant should not have read access to two or more banks or two or more oil companies.

The Chinese Wall policy has a mix of free choice and mandated restrictions. So long as a

consultant has not yet been exposed to any company information about banks, that consultant has the potential to read information about any bank. The moment this consultant reads, say, bank A information, thereafter that consultant is to be denied read access to all other banks. The free choice of selecting the first company to read in a COI class can be exercised once and is then forever gone (or at least gone for a sufficient length of time to avoid conflict of interest).

So long as we have focussed on read access the Chinese Wall policy has been easy to state and understand. When we turn to write access the situation becomes more complicated and subtle. This is the usual case with confidentiality policies. For example, the simple-security rule of the well-known Bell-LaPadula model [1] is similarly intuitive and straightforward whereas the $\star$-property (which prohibits write down) is more subtle.

In computer security it is easy to confuse the threat from malicious users with the threat from malicious subjects. In the Bell-LaPadula model, mandatory controls on write access are imposed to prevent Trojan Horse infected subjects from leaking information contrary to the system policy. These controls do not address the threat of malicious human users. It should always be kept in mind that a malicious user can compromise information confidentiality by employing communication means outside of the computer system. Thus John as a human being cleared to top-secret is nevertheless able to write and publish unclassified documents. This is because John is trusted not to leak top-secret information in his unclassified writings. On the other hand malicious subjects executing with John's top-secret privileges can leak top-secret information if not constrained by the $\star$-property.

In much the same way a computer system cannot solve the problem of a malicious consultant. A determined consultant can leak damaging confidential information about a company to, say, the Wall Street Journal by means of a telephone call. Similarly, a consultant can provide insider company information directly to its competitors or share this information with other consultants. Just as our top-secret user John is trusted not to divulge secrets, so must our consultants be trusted as individuals not to break Chinese Walls.

4 THE BREWER-NASH MODEL

We now consider the Brewer-Nash model for the Chinese Wall policy. In this model data is viewed as consisting of *objects* each of which belongs to a *company dataset*. The company datasets are categorized into conflict of interest (COI) classes, along the lines of 1.

The Brewer-Nash model does not distinguish users, principals and subjects. It uses the single concept of subject for all three notions. This leads them to propose the following mandatory rules.

1. *BN Read Rule*: Subject S can read object O only if
   - O is in the same company dataset as some object previously read by S (i.e., O is within the wall), or
   - O belongs to a COI class within which S has not read any object (i.e., O is outside the wall).
2. *BN Write Rule*: Subject S can write object O only if
   - S can read O by the BN read rule, and
   - no object can be read which is in a different company dataset to the one for which write access is requested.

We have called these the BN read rule and BN write rule for ease of reference. They are analogous to the simple-security and $\star$-properties of the Bell-LaPadula model.

The BN read rule conveys the dynamic aspect of the Chinese Wall policy. This rule clearly applies to the human users, viz., the consultants, in the system. Since the Brewer-Nash model does not distinguish between users and subjects, this rule is also applied to all subjects in the system.

The BN write rule is brought in to prevent Trojan Horse laden subjects from breaching the Chinese Walls. To see its motivation consider that consultant John has read access to Bank A objects and Oil Company OC objects, and that consultant Jane has read access to Bank B objects and Oil Company OC objects. Individually John and Jane are in compliance with the Chinese Wall policy. Now suppose John is allowed write access to OC objects. A Trojan Horse infected subject running with John's privileges can thereby transfer information from Bank A objects to OC objects. These OC objects can be read by subjects running on behalf of Jane, who then has read access to information about Bank A and Bank B.[5]

The BN write rule is successful in preventing such information leakage by Trojan Horses. However, it does so at an unacceptable cost. It is easy to see that the BN write rule has the following implication.

- A subject which has read objects from two or more company datasets cannot write at all.
- A subject which has read objects from exactly one company dataset can write to that dataset.

These implications are clearly unacceptable (if the computer system is to be used for something more than a read-only repository of confidential information). Under this regime a consultant can work effectively so long as he or she is assigned to exactly one company. The moment the consultant is assigned to a second company, he or she will be unable to write *any* information into the system.

Fortunately these implications are not inherent in the Chinese Wall policy. They are rather a consequence of the Brewer-Nash model's failure to distinguish rules applied to users from rules applied to subjects. The key observation is that we can live with the implications listed above with respect to subjects, but not with respect to users. In particular, limiting every subject to reading and writing a single company dataset is an acceptable restriction. Thus, any subject executing on behalf of John should either be able to read and write Bank A objects, or read and write Oil Company OC objects. John as a human being is, however allowed to read and write both Bank A and Oil Company OC objects. For that matter, John is also allowed to read and write public objects. However, he is not allowed to do all of these actions using the same subject.

5 A LATTICE INTERPRETATION

In this section we provide a lattice-based interpretation of the Chinese Wall policy. It was shown by Denning [3] that information flow policies in general require that objects be labeled with a lattice structure. Denning's result is derived from the following axioms.

1. Information flow is reflexive, transitive and symmetric.
2. There is a lowest class of information which is allowed to flow into all other classes.
3. For any two classes of information A and B there is a class C which is the least upper bound of A and B (i.e., (i) information from both A and B can flow to C, and (ii) for all classes D such that information can flow from both A and B it is the case that information can flow from C to D).

[5] Note that Computer Security cannot do anything to prevent John and Jane from exchanging Bank A and Bank B information outside of the computer system. But in such an exchange John and Jane are accomplices. In the example given here John is not an accomplice but rather an unwitting victim of a Trojan Horse.

These axioms are generally accepted as being very reasonable.[6] Now there is nothing in the Chinese Wall policy that is contrary to these axioms. We will bear out this claim by showing how we can construct a lattice structure for the Chinese Wall policy. We do so by defining a number of axioms below.

5.1 The Lattice Structure for Chinese Walls

Let us begin by introducing the conflict of interest classes and companies.

A1. There are n conflict of interest classes: COI_1, COI_2, ..., COI_n.

A2. $COI_i = \{1, 2, \ldots, m_i\}$, for $i = 1, 2, \ldots n$, i.e., each conflict of interest class COI_i consists of m_i companies.

In other words there are n conflict of interest classes, each of which contains some number of companies as visually depicted in figure 1.

We propose to label each object in the system with the companies from which it contains information. Thus an object which contains information from Bank A and Oil Company OC is labeled {Bank A, Oil Company OC}. Labels such as {Bank A, Bank B, Oil Company OC} are clearly contrary to the Chinese Wall policy. We prohibit such labels in our system by defining a security label as an n-element vector $[i_1, i_2, \ldots, i_n]$, where each $i_k \in COI_k$ or $i_k = \perp$.

An object labeled $[i_1, i_2, \ldots, i_n]$ is interpreted as signifying that it contains information from company i_1 of COI_1, company i_2 of COI_2 and so on. When an element of the vector is $\perp$ rather than an integer, it means that the object has no information from any company in the corresponding conflict of interest class. For example, an object which contains information only from company 4 in COI_3 will be labeled with the vector $[\perp, \perp, 4, \perp, \ldots, \perp]$, i.e., all elements other than the third one will be $\perp$. Similarly, an object which contains information from company 7 in COI_2 and company 5 in COI_4 will be labeled with the vector $[\perp, 7, \perp, 5, \perp, \ldots, \perp]$.

This leads us to the following definition for the set of labels.

A3. $LABELS = \{[i_1, i_2, \ldots, i_n] | i_1 \in COI'_1, i_2 \in COI'_2, \ldots, i_n \in COI'_n\}$ where $COI'_i = COI_i \cup \{\perp\}$

Note that the label which has all $\perp$ elements naturally corresponds to public information. There is, however, no naturally occurring system high label (in fact such a label is contrary to the Chinese Wall policy). In order to complete the lattice we introduce a special label for system high (which we will not assign to any subject in the system).

A4. $EXTLABELS = LABELS \cup \{SYSHIGH\}$

Next we define the dominance relation among labels as follows, where the notation $l_1[i_k]$ denotes the i_k-th element of label l_1.

A5. $(\forall l_1, l_2 \in LABELS)[l_1 \geq l_2 \Leftrightarrow (\forall i_k = 1, \ldots, n)[l_1[i_k] = l_2[i_k] \vee l_2[i_k] = \perp]]$

In other words, l_1 dominates l_2 provided that l_1 and l_2 agree wherever $l_2 \neq \perp$. For example $[1, 3, 2] \geq [1, 3, \perp]$, $[1, 3, 1] \geq [\perp, \perp, 1]$ while $[1, 3, 2]$ and $[1, 2, 3]$ are incomparable. Note that every label dominates the system low label which consists of all $\perp$ elements. To account for the special system high label we have the following axiom.

A6. $(\forall l \in EXTLABELS)[SYSHIGH \geq l]$

[6] Some researchers have tried to relax them further, for instance by dropping the transitive requirement on information flow, but in the main the security community has accepted these.

To complete the lattice structure it remains to define the least upper bound operator. In order to do so we introduce the following notion.

A7. $l_1, l_2 \in LABELS$ are compatible if and only if for all $k = 1, \ldots, n$, $l_1[i_k] = l_2[i_k] \vee l_1[i_k] = \perp \vee l_2[i_k] = \perp$

In other words, two label are compatible if wherever they disagree at least one of them is $\perp$. Note that if $l_1 \geq l_2$ then l_1 and l_2 are compatible. Labels which are incomparable with respect to the dominance relation may or may not compatible, e.g., $[1, 3, 2]$ and $[1, 2, 3]$ are incompatible while $[1, \perp, 2]$ and $[1, 2, \perp]$ are compatible.

Incompatible labels cannot be legitimately combined under the Chinese Wall policy. This is expressed by the following axiom.

A8. If l_1 is incompatible with l_2 then $lub(l_1, l_2) = SYSHIGH$

For compatible labels the least upper bound is computed as follows.

A9. if l_1 is compatible with l_2 then $lub(l_1, l_2) = l_3$ where $l_3[i_k] = \begin{cases} l_1[i_k] & \text{if } l_i[i_k] \neq \perp \\ l_2[i_k] & \text{otherwise} \end{cases}$

For example, the least upper bound of $[1, \perp, 2]$ and $[1, 2, \perp]$ is $[1, 2, 2]$. Finally to complete the definition with respect to the special system high label, we have the following axiom

A10. $(\forall l \in EXTLABELS)[lub(SYSHIGH, l) = SYSHIGH]$

It is easy to verify that the axioms **A1** to **A10** define a lattice on the set of labels $EXTLABELS$ with dominance relation $\geq$. Information flow occurs in the direction opposite to the dominance relation and is obviously reflexive, transitive and symmetric. The required system low class is identified by the label consisting of all $\perp$ elements, and the least upper bound operator has been defined.

Figure 2 shows a lattice with two conflict of interest classes, each with two companies in it. The lattice is shown by its Hasse diagram, in which the dominance relation goes from top to bottom with transitive and reflexive edges omitted.

5.2 Chinese Wall Model

Given this lattice structure we have developed, let us see how we can solve the Chinese Wall problem. To be concrete we describe our solution in terms of the specific lattice of figure 2. The solution is, however, completely general and applies to any size Chinese Wall lattice.

We require every object in the system to be labeled by one of the labels in figure 2. Public objects are labeled $[\perp, \perp]$. Objects with company information from a single company are labeled as follows:

- $[1, \perp]$: objects with information for company 1 in COI_1.
- $[2, \perp]$: objects with information for company 2 in COI_1.
- $[\perp, 1]$: objects with information for company 1 in COI_2.
- $[\perp, 2]$: objects with information for company 2 in COI_2.

Objects with company information from more than one company (without violation of Chinese Walls) are labeled as follows:

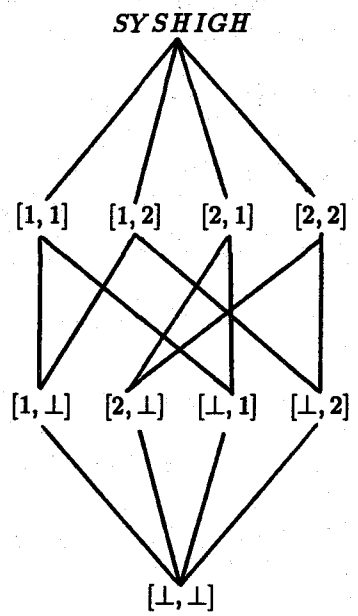


Figure 2: Example of a Chinese Wall Lattice

- [1, 1]: objects with information for company 1 in COI_1 and company 1 in COI_2.
- [1, 2]: objects with information for company 1 in COI_1 and company 2 in COI_2.
- [2, 1]: objects with information for company 2 in COI_1 and company 1 in COI_2.
- [2, 2]: objects with information for company 2 in COI_2 and company 2 in COI_2.

Objects labeled *SYSHIGH* violate the Chinese Wall policy, in that they can combine information from any subset of the companies. These objects are inaccessible in the system (and therefore might as well not exist).

Now let us consider labels on users, principals and subjects. We treat the label of a user as a high-water mark which can float up in the lattice but not down. A newly enrolled user in the system is assigned the label $[\bot, \bot]$.[7] As the user reads various company information the user's label floats up in the lattice.[8] For example, by reading information about company 1 in conflict of interest class 1 the user's label is modified to $[1, \bot]$. Reading information about company 2 in conflict of interest class 2 further modifies the user's label to $[1, 2]$.

This floating up of a user's label is allowed, so long as the label does not float up to *SYSHIGH*. Operations which would force the user's label to *SYSHIGH* are thereby prohibited. The ability

[7] This assumes that the user is entering the system with a "clean slate." A user who has had prior exposure to company information in some other system should enter with an appropriate label reflecting the extent of this prior exposure.

[8] The exact manner in which a user's label is allowed to float up is an issue of implementation. If the users have complete freedom in this respect, the proposed read access could be specified at the time of login. The system could then create a suitable principal for that user session. On the other hand one might constrain this by discretionary access controls which we have ignored in this paper. For instance, a user may be allowed to read only that company information which the user's boss assigns him or her to. In this case the float up of a user's label is effectively done by some other user. Full consideration of such discretionary policies and their interplay with the mandatory policy, would require a model such as the Typed Access Matrix [7].

to float a user's label upwards[9] addresses the dynamic requirement of the Chinese Wall policy. The floating label keeps track of a user's read operations in the system.

With each user we associate a set of principals, one at each label dominated by the user's label. Thus, if Jane as a user has the label $[1, 1]$, she has the following principals associated with her: Jane.$[1, 1]$, Jane.$[1, \perp]$, Jane.$[\perp, 1]$ and Jane.$[\perp, \perp]$. Each of these corresponds to the label with which she wishes to log in on a given session. These principals have fixed labels which do not change. The floating up of a user's label corresponds to creation of one or more new principals for that user. For example, when Jane had the label $[1, \perp]$, she had only two principals associated with her, viz., Jane.$[1, \perp]$ and Jane.$[\perp, \perp]$. When Jane's label floated up to $[1, 1]$, she acquired two new principals Jane.$[1, 1]$ and Jane.$[\perp, 1]$. This floating up of Jane's label is achieved by Jane's directive to the system. The system will allow this action only if the float up is to some label strictly below *SYSHIGH*.

Each principal has a fixed label. Every subject created by that principal inherits that label. Thus, all activity in the system initiated by Jane.$[1, \perp]$ will be carried out by subjects with the label $[1, \perp]$. The label of a subject is determined by the label of the principal who creates that subject. A subject's label remains fixed for the life of that subject.

All read and write operations in the system are carried out by subjects. These subjects are constrained by the familiar simple-security and $\star$-properties of the Bell-LaPadula model. That is a subject can only read objects whose labels are dominated by the subject's label, and can only write objects whose labels dominate the subject's label.

Now suppose that Jane logs in as the principal $[1, \perp]$. All subjects created during that session will inherit the label $[1, \perp]$. This will allow these subjects to read public objects labeled $[\perp, \perp]$, to read and write company objects labeled $[1, \perp]$, and write[10] objects with labels $[1, 1]$, $[1, 2]$ and *SYSHIGH*.

6 CONCLUSION

In this paper we have given a lattice interpretation of the Chinese Wall policy of Brewer and Nash [2]. In doing so we have disputed[11] their claim that the Chinese Wall policy "cannot be correctly represented by a Bell-LaPadula model." We have also shown that the Brewer-Nash model is too restrictive to be employed in practice, since it essentially prohibits consultants from adding new information into the system. By maintaining a careful distinction between users, principals and subjects, we developed a model for the Chinese Wall policy which addresses threats from Trojan Horse infected programs and retains the ability of consultants to write information into the company datasets they are analyzing. Our paper demonstrates the vital importance of distinguishing security policy as applied to human users versus security policy as applied to computer subjects.

The lattice model we have developed for the Chinese Wall policy uses the Bell-LaPadula simple-security and $\star$-properties. In this sense it is consistent with the Orange Book [4]. However, the structure of our security labels departs from the conventional military and government sector (with their hierarchical and non-hierarchical components). A system built to Orange Book criteria can be used to enforce Chinese Walls, provided there is some flexibility in the structure of the labels in the system.

[9] This float upwards does not present the security problems with changing labels discussed in [5]. This is due to the upward floating or high-water mark nature of our user labels.

[10] As is often done in multilevel secure database systems, we can prohibit this "write up" if we so choose.

[11] Although, see footnote 2 earlier in the paper.

References

[1] Bell, D.E. and LaPadula, L.J. "Secure Computer Systems: Unified Exposition and Multics Interpretation." MTR-2997, Mitre, Bedford, Massachusetts (1975).

[2] Brewer, D.F.C and Nash, M.J. "The Chinese Wall Security Policy." *IEEE Symposium on Security and Privacy*, 215-228 (1989).

[3] Denning, D.E. "A Lattice Model of Secure Information Flow." *Communications of ACM* 19(5):236-243 (1976).

[4] Department of Defense National Computer Security Center. *Department of Defense Trusted Computer Systems Evaluation Criteria.* DoD 5200.28-STD, (1985).

[5] McLean, J. "Reasoning About Security Models." *IEEE Symposium on Security and Privacy*, 123-131 (1987).

[6] Saltzer, J.H. and Schroeder, M.D. "The Protection of Information in Computer Systems." *Proceedings of IEEE* 63(9):1278-1308 (1975).

[7] Sandhu, R.S. "The Typed Access Matrix Model." *Proc. IEEE Symposium on Research in Security and Privacy*, Oakland, California, May 1992, pages 122-136.

A Local Area Network Security Architecture

Lisa J. Carnahan
National Institute of Standards and Technology
A216, Bldg. 225, Gaithersburg, MD, 20899

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to describe a process that can be used to improve the security of a local area network (LAN). This process is a risk-based approach based on perceived threats and vulnerabilities and considerations of security services and security mechanisms. A LAN security architecture is described that discusses threats and vulnerabilities that should be examined, as well as security services and mechanisms that should be considered. Appropriate LAN security should aim to achieve the following goals:

- Maintain the confidentiality of data as it is transmitted, stored or processed on a LAN;
- Maintain the integrity of data as it is transmitted, stored or processed on a LAN;
- Maintain the availability of data stored on a LAN, as well as the ability to process the data in a timely fashion;
- Ensure the identity of the sender and receiver of a message;
- Maintain the ability to transmit data in a timely fashion.

The process described in this paper has been applied to a LAN with an agency of the federal government. A description and outcome of this application is described in [1].

1.2 LAN Definition

A LAN is defined in general terms as "a network that is deployed in small geographic areas such as an office complex, building, or campus. Typically, a LAN is owned, operated, and managed locally rather than by a common carrier." [2] A LAN usually, through a common network operating system, connects servers, workstations, printers, and mass storage devices, enabling users to share the resources and functionality provided by a LAN. The types of applications provided by a LAN generally include distributed file storing, remote computing, and messaging.[3]

- *Distributed file storing provides users transparent access to part of the mass storage of a remote server.* Distributed file storing provides capabilities such as remote filing and remote printing. Remote filing allows users to access, retrieve, and store files. Generally remote filing is provided by allowing a user to attach to part of a remote mass storage device (a file server) as though it were connected directly. This virtual disk is then used as though it were a disk drive local to the workstation. Remote printing allows users to print to any printer attached to any component on the LAN; allowing users to utilize (and share the cost of) high quality printers, and allow ongoing local processing.

- *Remote computing refers to the concept of running an application or applications on remote components.* Remote computing allows users to remotely login to another component on the LAN, or remotely execute an application that resides on another component. Remote computing also allows a user to remotely run an application on one or more components, while having the appearance, to the user, of running locally.[2] The ability to run an application on one or more components allows the user to utilize the processing power of LAN as a whole.

- *Messaging applications are associated with mail and conferencing capabilities.* Electronic mail has been one of the most widely used capabilities available on computer systems and across networks. A conferencing capability allows users to actively communicate to each other, analogous to the telephone.[3]

1.3 The LAN Security Problem

The advantages of using a LAN were discussed in the previous section. However, with these advantages in functionality come added risk to the data that is processed, stored and communicated. Other areas of concern that can increase risk include (1) poor LAN management and security policies, (2) lack of training for LAN usage and security, (3) poor protection mechanisms in the workstation environment, and (4) the use of efficient LAN protocols. These additional concerns are mentioned here for completeness and will not be discussed in detail.

File servers can control users' accesses to various parts of the file system. This is usually done by allowing a user to attach a certain file system (or directory) to the user's workstation, to be used as a local disk. However, two potential problems arise with this. First, the server may only provide access protection to the directory level, so that a user granted access to a directory has access to all files contained in that directory. The second problem is caused by the lack of protection mechanisms on the local workstation. For example, a personal computer (PC) may provide minimal or no protection of the information stored on it. A user that copies a file from the server to the local drive on the PC loses the protection afforded the file when it was stored on the server. For some types of information this may be acceptable. However, other types of information may require more stringent protections. This requirement then focuses on the need for controls in the workstation environment.

Distributed computing must be controlled so that only authorized users may access remote components and remote applications. Components must be able to authenticate

remote users who request services or applications. These requests may also call for the local and remote commponents to authenticate to each other. The inability to authenticate can lead to unauthorized users being granted access to remote components and applications.

Network protocols and topologies that do not provide a direct, point to point path from sender to receiver, should be recognized as a concern. Standard topologies and protocols used today demand that messages pass through many nodes to get to the destination. This is much cheaper, and easier to maintain than providing a direct physical path for every machine to every machine. (In large LANs, direct paths may be infeasible.) Making the information that is transmitted unintelligible becomes apparent, considering the ease at which an intruder can listen to the traffic as it is transmitted across the LAN.

Messaging services add additional risks to information that is stored on a server as well as in transit. Electronic mail that is transmitted over the LAN could easily be captured, and perhaps altered and retransmitted, affecting both the confidentiality and integrity of the message.

The use of personal computers in the LAN environment can also add risk to the LAN. In general, PCs have a lack of strong security mechanisms for authenticating users, controlling access to files, auditing, etc. In many cases, the protection afforded information that is stored and processed on a LAN server does not follow the information when it is sent locally to a PC.

The solution to providing adequate LAN security is to provide the proper combination of security policies and procedures, technical controls, user training and awareness, and contingency planning. While all of these areas are critical for providing adequate protection, the focus of this document is on the technical controls that can be employed. These controls can be defined by a LAN security architecture. This architecture defines common threats to the LAN, as well as the needed technical controls.

1.4 LAN Security Architecture Definition

In this document a LAN security architecture (which describes the security functionalities of the LAN) is defined by the relationships between threats, vulnerabilities, security services and security mechanisms.

A threat can be any person, object, event, or idea that, if realized, could potentially cause damage to the LAN. Threats can be malicious, such as the intentional modification of sensitive information, or can be accidental, such as an error in a calculation, or the accidental deletion of a file. Threats can also be acts of nature, i.e. electrical spikes, water damage, lightning, etc.

Vulnerabilities are weaknesses in a LAN that can be exploited by a threat. For example, unauthorized access (the threat) to the LAN could occur by exploiting a vulnerability such as a poor password choice made by a user. Reducing or eliminating the vulnerabilities of the LAN can reduce or eliminate the risk of the threats to the LAN. The use of a robust password generator may reduce the chance that a user will choose a poor password, and thus reduce the threat of unauthorized LAN access.

A security service is the collection of security mechanisms, procedures, etc. that are implemented on a LAN to protect the LAN from threats. The identification and authentication service could be designed to help protect the LAN from unauthorized LAN access by requiring that a user identify him/herself, and provide something else that verifies his/her identity. A security service is only as robust as the mechanisms, procedures, etc. that make up that service.

Security mechanisms are the controls implemented to provide the security services that are needed to protect the LAN. For example, a token based authentication system (which requires that the user be in possession of a required token) may be the mechanism implemented to provide the identification and authentication service.

Using the relationships defined above, the *LAN security architecture* describes the security functionality by specifically:

- *defining security threats and associated vulnerabilities* to the LAN,
- *listing the security services and associated mechanisms* that can provide protection,
- *depicting the mapping of the threats and vulnerabilities to the required security services and mechanisms.*

Therefore a LAN security architecture will consist of a set of components where each component contains four elements. These elements are the threat, the associated vulnerability, the security service that helps provide protection from the threat, and the implemented security mechanism(s) that make up the security service.

1.5 Priorities for LAN Security

A risk analysis can be used to determine the appropriate level of protection required for a LAN. There are many methods that can be utilized to perform a risk analysis. This document suggests a risk analysis process as follows:

(1) Using the LAN security architecture that will be presented in Section 2, determine the level of risk associated with each of the threats and the vulnerabilities that exist;
(2) For components where the threat is associated with an unacceptable level of risk, determine the security services and mechanisms that would be appropriate to reduce this risk to the LAN. This is done based on a cost justification basis. (A detailed discussion on performing the risk analysis is presented in Section 3, 'Determining Priorities for LAN Security'.)

The result of the risk analysis is a list of specific components from the presented LAN security architecture that are determined as significant (based on the level of risk and the cost to reduce that risk) for a given LAN. These components are called the 'priorities for LAN security' and

are those components that need to be addressed in order to obtain an acceptable level of assurance for the security of the LAN. Specifically *the priorities for LAN security consist of an ordered list of specific architecture components that delineate both a threat with an unacceptable risk and mechanism(s) with a justifiable implementation cost.* The ordering of the list is determined by a ratio of the risk of the threat and the cost to reduce the threat. Those components where the greatest risk is reduced by the least cost are ranked higher in the list.

2 THREATS, VULNERABILITIES, SECURITY SERVICES & MECHANISMS

This section is composed of two parts. The first part discusses threats and related vulnerabilities. The second part of this section discusses LAN security services and the possible mechanisms that can be implemented to provide these services. This section refers the reader to Tables 1 and 2, and Figure 1. Table 1 - Threats and Related Vulnerabilities provides a listing of specific vulnerabilities that could be exploited by the threats discussed here. Table 2 - Security Services & Related Mechanisms presents possible security mechanisms that could be incorporated into the security services that are discussed. Figure 1 - Relating Threats and Security Services provides a matrix to show the relationships between the threats and security services.

2.1 Threats and Vulnerabilities

The following paragraphs discuss the threats and vulnerabilities that are incorporated into the LAN security architecture. The threats that will be discussed are:

- **Unauthorized LAN access**
- **Unauthorized access to LAN resources**
- **Compromise of data**
- **Unauthorized Modification to data**
- **Compromise of LAN traffic**
- **Modification to LAN traffic**
- **Spoofing of LAN traffic**
- **Disruption of LAN functionalities**

2.1.1 Unauthorized LAN Access

LANs provide file sharing, printer sharing, storage sharing, etc. Because resources are shared and not utilized solely by one individual, there is a need for control of the resources and accountability for use of the resources. *Unauthorized LAN access occurs when someone, who is not authorized to use the LAN or to have access to the files and resources available on the LAN, gains access to the LAN (usually by acting as a legitimate user of the LAN).* Two common methods used to gain unauthorized access are general password guessing, and password capturing. General password guessing is not a new means of unauthorized access. However, with LANs having large repositories of data, software, etc., (compared to the amount of information stored on a single-user system) the consequences of this threat could be extreme. Password capturing is a process in which a legitimate user may unknowingly reveal his/her login id and password. This can be done by using the login

UNAUTHORIZED LAN ACCESS
• Lack of/weak identification and authentication (I&A) mechanism • Poorly managed open systems • Poor password management • Trojan horse/back door programs • Unprotected modem use • Lack of I&A scheme on PCs • Poor physical control of LAN devices
UNAUTHORIZED ACCESS TO LAN RESOURCES
• Use of lenient system default permissions • Improper use of LAN manager privileges • Lack of/poorly managed access control • Lack of access control for data on PCs
COMPROMISE OF DATA & SOFTWARE
• Lack of encryption for sensitive data • Monitors & printout stations placed in high traffic areas • Backup copies of information and data not secured
UNAUTHORIZED MODIFICATION TO DATA & SOFTWARE
• Lenient write/modify access rights • Undetected changes to software • Lack of cryptographic checksum on sensitive data • Privilege mechanism allowing excessive write permission
COMPROMISE OF LAN TRAFFIC
• Inadequate physical protection of LAN devices • Use of broadcast protocols • Transmitting plaintext data
UNAUTHORIZED MODIFICATION TO LAN TRAFFIC
• Lack of cryptographic checksum use
SPOOFING OF LAN TRAFFIC
• Transmitting plaintext • Lack of date/time stamp • Lack of message authentication code or digital signature • Lack of real-time verification mechanism
DISRUPTION OF LAN FUNCTIONALITIES
• Inability to detect unusual traffic patterns • Inability to reroute traffic, handle h/w, s/w failures • Allowing for single point of failure

Table I - Threats and Related Vulnerabilities

program as a trojan horse that can reveal a user's login id and password to the potential intruder. Capturing an unencrypted login id and password as it is transmitted across the LAN is another method used to gain access.

2.1.2 Unauthorized Access to LAN Resources

One of the benefits of a LAN is that many resources are readily available to many users, rather than each user having limited dedicated resources. However, not all resources need to be made available to each user. To prevent compromising the security of the resource, (i.e., corrupting the resource, or lessening the availability of the resource) only those who require use of the resource should be permitted to utilize that resource. *Unauthorized access occurs when a user, legitimate or unauthorized, accesses a resource that he/she is not permitted to use.* Unauthorized access may occur simply because the access rights assigned to the resource are not assigned properly. However, unauthorized access may also occur because the access control mechanism, or the privilege mechanism is not granular enough. In these cases, the only way to grant the needed access rights or privileges is to grant more access than is needed, or more privileges than are needed.

2.1.3 Compromise of LAN Data

As LANs are utilized throughout an agency or department, some of the data stored, processed or transmitted throughout the LAN may require some level of confidentiality. *The compromise of LAN data occurs when an individual, who should not be privy to the data, breaks the confidentiality of the data by accessing it and comprehending it.* This can occur by someone gaining access to information that is not encrypted, or by viewing monitors or printouts of the information.

2.1.4 Unauthorized Modification of Data and Software

Because LAN users share data and applications, changes to these resources must be controlled. *Unauthorized modification of data or software occurs when unauthorized changes (additions, deletions or modifications) are made to a file or program.*

When undetected modifications to data are present for long periods of time, the modified data may be spread throughout the network, possibly corrupting databases, spreadsheet calculations, and other various application data. This can damage the integrity of most application information.

When undetected software changes are made, all system software can become suspect, warranting a thorough review (and perhaps reinstallation) of all related software and applications. These unauthorized changes can be made in simple command programs (for example in PC batch files), in utility programs used on multi-user systems, in major application programs, or any other type of software. They can be made by unauthorized outsiders, as well as those who are authorized to make software changes (although the changes they make are not authorized). These changes can divert information (or copies of the information) to other destinations, corrupt the data as it is processed, or impact the availability of system or network services.

2.1.5 Compromise of LAN Traffic

The compromise of LAN traffic occurs when someone who is unauthorized reads, or otherwise obtains, information as it travels across the LAN medium. LAN traffic can be compromised by physically tapping the network cable (or listening to traffic that is transmitted through the air) or capturing broadcast traffic based on an address. Many users realize the importance of confidential information when it is stored on their workstations or servers; however, it is also important to maintain that confidentiality as the information travels through the LAN. Information that can be compromised in this way includes system and user names, passwords, electronic mail messages, application data, etc. For example even though passwords may be in an encrypted form when stored on a system, they can be captured in plaintext as they are sent from a workstation or PC to a file server. Electronic mail message files, which usually have very strict access rights when stored on a system, are often sent in plaintext, making them an easy target for capturing.

2.1.6 Modification to LAN traffic

Data that is transmitted over a LAN should not be altered in an unauthorized manner as a result of that transmission, either by the LAN itself, or by an intruder. LAN users should be able to have a reasonable expectation that the message sent, is received unmodified. *A modification occurs when a change is made to any part of the message including the contents and addressing information.*

2.1.7 Spoofing of LAN Traffic

Messages transmitted over the LAN need to contain addressing information that reports the sending address of the message and the receiving address of the message (along with other pieces of information). *Spoofing of LAN traffic involves (1) the ability to receive a message by masquerading as the legitimate receiving destination, or (2) masquerading as the sending machine and sending a message to a destination.* To masquerade as a receiving machine, the LAN must be fooled into believing that the destination address is the legitimate address of the machine. (Receiving LAN traffic can also be done just by listening to messages as they are broadcast to all nodes.) Masquerading as the sending machine to deceive a receiver into believing the message was legitimately sent can be done by spoofing the address, or by means of a playback. A playback involves capturing a session between a sender and receiver, and then retransmitting that message (either with the header only, and new message contents, or the whole message).

2.1.8 Disruption of LAN Functionalities

A LAN is a tool, used by an organization, to share information and transmit it from one location to another.

This need is satisfied by LAN functionalities such as those described in Section 1.2, 'LAN Definition'. *A disruption of functionality occurs when the LAN cannot provide the needed functionality in an acceptable, timely manner.* A disruption can interrupt one type of functionality or many.

2.2 Security Services and Mechanisms

A security service is the collection of mechanisms, procedures, etc. that are implemented to help reduce the risk of associated threats. For example, the identification and authentication service protects the network from the unauthorized user threat. Some services help provide protection from threats, while other services provide for detection of the threat occurrence. An example of this would be a logging or monitoring service. The following services will be discussed in this section:

- **Identification and authentication**
- **Access control**
- **Data confidentiality**
- **Data integrity**
- **LAN message confidentiality**
- **LAN message integrity**
- **Non-repudiation**
- **Logging and Monitoring**

When determining the priorities for LAN security, the services should be viewed as providing a layered approach. While most services can stand alone and provide protection from a specific threat, using as many as possible in conjunction strengthens them all.

2.2.1 Identification and Authentication

Users who access workstations, servers, etc. on a LAN may need to be identified and authenticated to each of those systems. Identification requires the user to be known by the system in some manner. This is usually based on an assigned userid. However the system cannot trust the validity that the user is in fact, who he/she claims to be, without being authenticated. The authentication is done by having the user supply something that only the user has, such as a token, something the user knows, such as a password, or something that makes the user unique, such as a fingerprint. The more of these that the user has to supply, the less the chances are that someone can masquerade as a legitimate user.

On most LANs, the identification and authentication mechanism is a userid/password scheme. However more LANS are implementing a mechanism where the user supplies a token (usually a smartcard) and a password. This means that the user must possess something (the token) and know something (the password) to gain access.

2.2.2 Access Control

This service protects against the unauthorized use of LAN resources, and can be provided by the use of access control mechanisms and privilege mechanisms. Most file servers and multi-user workstations provide this service to some extent. However, PCs which mount directories from the file servers usually do not. It is important to realize that no

IDENTIFICATION & AUTHENTICATION
• Identification and authentication (I&A) mechanism using passwords, smart cards/tokens, biometrics, some combination
• I&A mechanism used for all LAN devices
• Keyboard/workstation locking
• Password generator
• Termination of connection upon multiple login failures
• User restrictions to needed devices only
• Realtime user verification
ACCESS CONTROL
• Mechanism using permission bits, access control lists, user profiles, etc.
• Granular privilege mechanism
• Encryption for sensitive files
• Program execution based on access control and privilege
DATA CONFIDENTIALITY
• Encryption
• Use of partitions, screens, etc. to block screen view
• Protection for backup copies of data and software, printouts, etc.
• Appropriate access control settings
DATA INTEGRITY
• Message authentication codes on software and data
• Appropriate access control settings
• Granular privilege mechanism
• Virus detection software
• Workstations with no local storage, no software input device
LAN MESSAGE CONFIDENTIALITY
• Message encryption
• Point-to-point protocols
• LAN devices to limit/scramble broadcasting
• Physical protection of LAN medium
LAN MESSAGE INTEGRITY
• Use of message authentication codes
NON-REPUDIATION
• Use of public key digital signature
LOGGING AND MONITORING
• Logging of I&A information
• Logging of changes to access control information
• Logging the use of sensitive files & critical

Table II - Security Services & Related Mechanisms

matter how stringent the access control on a file server is, once the files are mounted as a logical disk on a PC, that security is no longer there. For this reason it may be important to try and incorporate this service on PCs to whatever extent possible.

Access control can be achieved by using discretionary access control or mandatory access control. Discretionary access control is the most common type of access control used by LANs. The basis of this kind of security is that an individual user, or program operating on the user's behalf is allowed to specify explicitly the types of access other users (or programs executing on their behalf) may have to information under the user's control. Discretionary security differs from mandatory security in that it implements the access control decisions of the user. Mandatory controls are driven by the results of a comparison between the user's trust level or clearance and the sensitivity designation of the information. [4, pg.2]

Most LAN access control mechanisms support access granularity to the level of acknowledging an owner, specified groups of users, and the world. Many LAN operating systems implement user profiles or access control lists to specify control for individual users. These mechanisms allow more flexibility in granting different accesses to different users (than the owner/group/world scheme), while providing more stringent access to the file. (It can prevent having to give a user more access than is necessary, a common problem with the three level approach.)

Privilege mechanisms enable authorized users to override the access permissions, or in some manner legally bypass some controls to perform a function, access a file, etc. An example of this may be that a user is granted a privilege to override read restrictions on all files in order to perform the backup function. The more granular the privileges that can be granted, the more control there is in not having to grant unnecessary privilege. For example, the user who has to perform the backup function does not need to have a write override privilege, but for privilege mechanisms that are less granular, this may occur.

2.2.3 Data Confidentiality

This service helps to protect data on workstations, file servers, etc. from unauthorized disclosure. This service can be provided by an encryption mechanism, often in conjunction with the access control service. In this way, if the access control mechanism is circumvented, the file may be accessed but the information is still protected by being in encrypted form. The use of an encryption mechanism can be very effective on PCs that do not provide an access control service.

2.2.4 Data Integrity

This service helps to protect data on workstations, file servers, etc. from unauthorized modification. The unauthorized modification can be intentional or accidental. This service can be provided by the use of cryptographic checksums, and very granular access control and privilege mechanisms.

The use of cryptographic checksums provide a modification detection capability. A Message Authentication Code (MAC), a type of cryptographic checksum, can protect against both accidental and intentional, but unauthorized, data modification. A MAC is initially calculated by applying a cryptographic algorithm and a secret value, called the key, to the data. The initial MAC is retained. The data is later verified by applying the cryptographic algorithm and the same secret key to the data to produce another MAC; this MAC is then compared to the initial MAC. If the two MACs are equal, then the data is considered authentic. Otherwise, an unauthorized modification is assumed. Any party trying to modify the data without knowing the key would not know how to calculate the appropriate MAC corresponding to the altered data[5,pp.1-2]. See [5] for more information regarding the use of MACs.

2.2.5 LAN Message Confidentiality

This service protects the information from compromise as it travels through the medium. This service is critical to most networks. For nondisclosable information, there must be a relatively high level of trust that the information is not readable to anyone other than the intended recipient. This means that either (1) only the intended user has access to the information, or (2) the information is unreadable to anyone else who gains access to the information.

It is very difficult to control access to network traffic as it traveling across the medium (unless all the wires are physically encased and protected, and the network is not a broadcast type of network). For most networks this is a realized and accepted problem. Therefore the mechanism of choice for this service involves some type of encryption, to make it unreadable to those who may capture it.

2.2.6 LAN Message Integrity

This service helps to ensure that a message is not altered, deleted or added to in any manner during transmission. Most of the techniques available today cannot prevent the modification of a message, but they can detect the modification of a message (unless the message is deleted altogether). Sending data across a LAN in encrypted form will not prevent the message from being altered; however, when the message is decrypted, in most cases it should be obvious that it was tampered with or that an attempted addition was made. A stronger approach than using simple encryption is to calculate a message authentication code (MAC) for the message. The MAC is calculated based on the contents of the message. After transmission another MAC is calculated on the contents of the received message. If the MAC associated with the message that was sent, is not the same as the MAC associated with the message that was received, then there is proof that the message received does not exactly match the message sent.

2.2.7 Non-repudiation

Non-repudiation ensures that the parties in a communication cannot deny having participated in all or part of the communication. When a major function of the LAN is electronic mail, this service becomes very important. This takes two forms (1) non-repudiation with proof of origin and (2) non-repudiation with proof of delivery. Non-repudiation with proof of origin gives the receiver confidence that the message indeed came from the named sender. Non-repudiation with proof of delivery gives the sender confidence that the message was delivered to the named receiver.

2.2.8 Logging and Monitoring

This service performs two functions. The first is the detection of the occurrence of a threat. (However, the detection does not occur in real time unless some type of real-time monitoring capability is utilized.) Depending on the extensiveness of the logging, the detected event should be traceable throughout the system. For example, when an intruder breaks into the system, the log should indicate who was logged on to the system at the time, all sensitive files that had failed accesses, all programs that had attempted executions, etc. It should also indicate sensitive files and programs that were successfully accessed in this time period. It may be important that all areas of the network (all workstations, fileservers, etc.) have some type of logging service.

The second function of this service is to provide system and network managers with statistics that indicate that systems and the network as a whole are functioning properly. This can be done by an audit mechanism that uses the log file as input and processes the file into meaningful information regarding system usage and security. A monitoring capability can also be used to detect LAN availability problems as they develop.

3 DETERMINING PRIORITIES FOR LAN SECURITY

A systematic approach should be utilized to determine appropriate LAN security measures. This section will describe a risk analysis method that can be used to determine appropriate security measures for existing LANS. This approach can be exercised for LANs that are in the development process as well. This approach uses the LAN security architecture described in Section 2, and describes a risk analysis process that can be used to determine the priorities for LAN security for a given LAN. The five step process begins with a data collection phase that stresses the need for detailing the physical and functional aspects of the LAN, as well as the importance of identifying and valuing all assets of the LAN. The next steps of the process address what harm could come to the LAN, the consequences of that harm to the assets, and what possible security measures could be taken to protect the LAN. The last step of the process involves implementing these

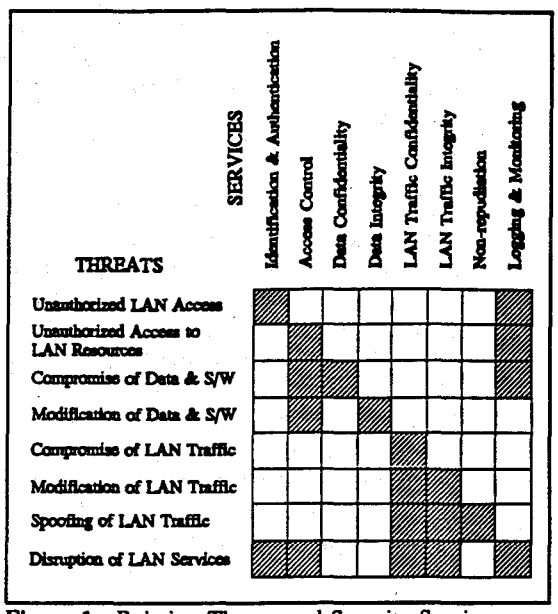


Figure 1 - Relating Threats and Security Services

measures, and testing them to ensure that the security is appropriate. The five steps for this process are:

1. *Define the LAN configuration,*
2. *Determine the LAN risks,*
3. *Select security services and security mechanisms,*
4. *Develop 'priorities for LAN security',*
5. *Implement and test security mechanisms.*

LAN security should not be addressed by one individual. It is important that the concerns and needs of the organization as a whole are addressed. This perspective can only be obtained by including parties from relevant areas of the organization, which minimally may include LAN management, organizational management, and security personnel.

STEP 1 DEFINE LAN CONFIGURATION

The first step in determining priorities for LAN security is to define all aspects of the LAN, and to determine all assets of the LAN. The goal of this step is twofold, the first is to have a detailed LAN configuration that indicates hardware ineorporated, major software applications used, significant information processed on the LAN, as well as how that information flows through the LAN. The second goal of this step is to identify and value the assets of the LAN. An asset is any part of the LAN considered to have value. Assets may include any piece of hardware, software, applications, data, etc. Assets then become those areas of the LAN that need to be protected. When developing the LAN configuration, the following aspects should be considered:

1. Hardware configuration - includes servers, workstations, PCs, peripheral devices, remote connections, cabling maps, bridges or gateway connections, etc.
2. Software configuration - includes server operating

systems, workstation and PC operating systems, the LAN operating system, major application software, software tools, LAN management tools, and software under development. This should also include the location of the software on the LAN, and from where it is commonly accessed.

3. Data - includes a meaningful typing of the data processed and communicated through the LAN, as well as the types of users who generally access the data. Indications of where the data is stored and processed, along with how the data flows through the LAN is important. Attention to the sensitivity of the data should also be considered.

In determining and valuing LAN assets, this process uses a qualitative valuation approach. The value of the asset is represented in the process in terms of the potential loss if a threat is realized. The loss value for the asset is calculated as a value between 1 and 3, meaning a 1 will indicate a low loss, a 2 will indicate a moderate loss, and a 3 will indicate a high loss.

After the LAN configuration is completed, and the assets are determined and valued, there should be a reasonably correct view of what the LAN consists of, and what areas of the LAN need to be protected. This leads to Step 2 - Determine Risk, which will indicate what can harm the LAN and how vulnerable the LAN is to realizing losses.

STEP 2 DETERMINE RISK

The question - What are the LAN assets that need protection? has been answered in the previous step. To answer the question - From what threats do the assets need protection?, an understanding of the threats and vulnerabilities needs to be developed. This understanding can be accomplished by performing a risk analysis. A risk analysis measures how vulnerable the LAN is to defined threats. The goal of this step is to determine the level of current security for the LAN, by determining the risk of the LAN to threats and vulnerabilities.

To begin the process of determining risk, consider the threats to the LAN, and the possible vulnerabilities of the LAN that could be exploited by those threats. Use the threat and vulnerability lists provided in the LAN security architecture to examine the LAN, however do not preclude other threats and vulnerabilities that may be discovered. Add these new threats and vulnerabilities to the threat/vulnerability lists. Any aspect of the LAN that was defined in step 1 to have value should be examined to determine what threats could potentially harm it. Particular attention should be made to detail the ways that these threats could occur. For example, unauthorized access may be from a login session playback, password cracking, the attachment of unauthorized equipment to the LAN, etc. These specifics provide more information in determining LAN vulnerabilities, which will provide more information in making determinations in later steps.

The risk analysis may uncover some vulnerabilities that can be corrected by improving LAN management and operational controls immediately. These improved controls will usually reduce the risk of the threat by some degree, until such time that more thorough improvements are planned and implemented.

Attention should be paid to existing LAN security controls to determine if they are not currently providing adequate protection, and thus become vulnerabilities. These controls may be technical, procedural, etc. For example, a LAN operating system may provide access control to the directory level, rather than the file level. For some users, the threat of compromise of information may be too great not to have file level protection. In this example, the access control provided could be considered a vulnerability.

As specific threats and related vulnerabilities are identified, a risk value needs to be associated with the threat. The risk associated with a threat is generally defined to be a function of the probability that the threat can occur, and the expected loss incurred given that the threat occurred. The risk can be calculated as follows:

risk = probability of threat occurring x loss incurred

The value estimated for loss is determined to be a value between 1 and 3. (This should have accomplished in Step 1 in conjunction with asset identification.) The probability of the threat occurring can also be normalized between 1 and 3, meaning a 1 will indicate a low probability, a 2 will indicate a moderate probability and a 3 will indicate a high probability. Therefore risk will be calculated as a number between 1 and 9 (actually the possibilities are 1,2,3,4,6 and 9), meaning a risk of 1 or 2 is considered a low risk, a risk of 3 or 4 would be a moderate risk, and a risk of 6 or 9 would be considered a high risk. For example, it could be considered that the loss of data may be valued at 3. The probability that a threat may occur to cause this loss of data may be estimated at 2. Therefore the calculation may be:

risk = 3 x 2 = 6 = HIGH

The levels of risk are now normalized (i.e. low, medium and high) and can be used to compare risks associated with each threat. Using an approach such as this is useful for many who are responsible for developing LAN security, however it does not preclude calculating risk by a different method.

To ensure that all identified risks and vulnerabilities are addressed, construct a list by prioritizing the threats based on the risk associated with each threat. Threats with the highest risk should be placed at the top, while threats with a lower risk value at the bottom. The vulnerabilities related to the threats should appear with the threats.

With a list of potential threats, vulnerabilities and related risks, an assessment of the current security situation for the LAN can be determined. Areas where there is adequate protection do not surface as contributing to the risk of the LAN, whereas those areas that have weaker protection do surface as needing attention. These are the areas that are considered in Step 3 - Security Service and Mechanism Selection, which analyzes potential security services and mechanisms in order to provide adequate protection.

STEP 3 SELECT SECURITY SERVICES & MECHANISMS

This step examines security services and mechanisms to determine those that would be appropriate to provide security to reduce the defined risks. Security services are the sum of mechanisms, procedures, etc. that are implemented on the LAN to provide protection. The goal of this step is to determine the possible security services and mechanisms needed, based on the risk information provided from the previous step. When deciding on services and mechanisms, the issue of funding the services and mechanisms should not be used to preclude including a specific service or mechanism. All feasible services and mechanisms should be considered and included in the process in this step (service and mechanism choices based on available funding are considered in Step 4 - Develop Priorities for LAN Security). This step is broken into four tasks.

TASK 1 - Consider the security services provided in the LAN security architecture. To determine if a specific security service is needed, use the matrix provided with the LAN security architecture to help in the consideration. Relate the threats defined in the previous step to the services that are shown to help reduce the risk of the threat. In most cases the need for a specific service should be readily apparent. If there is no risk to a certain threat (if existing mechanisms are adequate) then there is no need to apply additional mechanisms to the service that already exists.

TASK 2 - After the needed security services are determined, consider the list of security mechanisms for each service. For each security service selected, determine the candidate mechanisms that would best provide that service. The issue of available funding should not be factored in this decision. Using the threat/vulnerability relationships developed in the previous step, chose those mechanisms that could potentially reduce or eliminate the vulnerability, and thus the risk of the threat. In many cases, a threat/vulnerability relationship will have more than one candidate mechanism. Choosing the candidate mechanisms is a subjective process that will vary from one LAN implementation to another. Not every mechanism presented in the LAN security architecture is feasible for use in every LAN. In order for this process to be beneficial, some filtering of the mechanisms presented needs to be made during this step. Mechanisms should not be included in the list of candidate mechanisms if they can be discounted for a special reason (incompatibilities with existing configurations or mechanisms, policy issues, etc).

TASK 3 - The decision to use a certain mechanism will largely depend on the cost of the mechanism. Although the decision to implement a certain mechanism is made in the next step, the estimation of the cost will be made in this step. This cost is the amount needed to purchase or develop, and implement each of the mechanisms. The cost can be normalized in the same manner as was the value for potential loss incurred, that is a 1 will indicate a mechanism with a low cost, a 2 will indicate a mechanism with a moderate cost, and a 3 will indicate a mechanism with a high cost.

TASK 4 - In order to relate the threat/vulnerability relationships with the candidate security services and mechanisms, update the list of threats, vulnerabilities and associated risk, to show the relationship of these to the candidate security services, security mechanisms, and estimated costs. With the completion of this step, (and thus the risk analysis) the following should now be defined:

1. The threat/vulnerability relationships accompanied by the associated risk of the threat.
2. The proposed security services that could be used to adequately protect the LAN from threats.
3. The feasible security mechanisms (with cost estimates) that could be used to comprise the security services.

These constitute the input for the subsequent step used to determine those specific security mechanisms that should be implemented. The analysis that is performed in the next step can only be as solid as the information provided to it.

STEP 4 DEVELOP 'PRIORITIES FOR LAN SECURITY'

In this step a determination is made of which candidate security services and mechanisms will provide acceptable protection, given cost and other concerns. The goal of this step is to produce a prioritized list of security services and mechanisms that should be implemented in order to reduce perceived LAN risks and to protect the LAN adequately (called the priorities for LAN security).

The process used in this step to determine the proper services and mechanisms involves a comparison of the risk associated with a threat, and the cost estimated to implement a mechanism that will reduce the threat. This comparison is made for each component that was created in the preceding step. In some cases this process may not be straightforward. Other factors such as special concerns, requirements, policies, etc. may mandate that a specific mechanism be implemented, regardless of the cost. In some cases there may be mechanisms that reduce or eliminate more than one vulnerability (and thus reduce the risk of one or more threats). In these cases, it may be appropriate to group the LAN security architecture components together and recognize that a particular mechanism has the potential to reduce or eliminate more than one vulnerability (and the risk of more than one threat).

To calculate the risk/cost relationships use the risk value and the cost value associated with each threat/mechanism relationship and create a ratio of the risk to the cost (i.e. risk/cost). A ratio that is less than 1 will indicate that the cost of the mechanism is higher than the risk associated with the threat. This is generally not an acceptable situation (and may be hard to justify) but should not be automatically dismissed as a possibility. Consider that the risk value is a function of both the loss value and the probability value. One or both of these values may represent something so critical about the asset that the risk

value does not properly reflect the loss. Every LAN implementation has different security needs, and in certain cases, a threat/mechanism relationship with a value less than 1 may be warranted. Also, since these are estimates, something less than a 1 but close to a 1 may be reflecting the diffuculty of estimating. An additional column needs to be added to the list showing the risk/cost relationship.

To determine which components constitute priorities for LAN security, the components should be ranked based on their risk/cost values. Rank those with the highest risk/cost value first, since these reduce the most risk for the least cost. This process of ranking provides only a guideline for choosing appropriate mechanisms. Other factors may provide justification for ranking a component higher.

The goal of this step is to determine the appropriate security mechanisms to implement on the LAN by ranking the components based on the risk/cost information provided, and other considerations, if any. The priorities for LAN security suggest necessary mechanisms that need to be implemented to provide adequate LAN security. It is possible that some components (usually those with a lower ranking) may not be considered as a priority. This could be due to unjustifiable risk/cost ratios, certain organization policies, or other factors that make them infeasible. Components considered infeasible, for whatever reason, can be removed from the list. The remaining components of the prioritized list become the priorities for LAN security. The mechanisms that make up these components should then be implemented as funding becomes available.

With the completion of this step, the goal of the overall process has been met - that is, to determine the appropriate security measures needed to protect the LAN. These measures are referred to as the priorities for LAN security. The final step of this process is utilized to ensure that the mechanisms are implemented correctly, and that they provide the security they are supposed to provide.

STEP 5 IMPLEMENT AND TEST SECURITY MECHANISMS

Just as the mechanisms that constitute the priorities for LAN security were chosen using a systematic approach, so should the implementation of those mechanisms proceed in the same manner. The goal of this phase is to ensure that the security mechanisms are implemented correctly, are compatible with other LAN functionalities and security mechanisms, and that the security mechanisms meet the requirements of providing adequate security.

This step begins by developing a plan to implement the mechanisms. This plan should consider factors such as the timeliness required to reduce risk, available funding, users' learning curve, etc. A testing schedule for each mechanism should be incorporated into this plan. This schedule should show how each mechanism interacts with other mechanisms (these may be security mechanisms or mechanisms of some other functionality). The expected results (or the assumption of no conflict) of the interaction should be detailed. It should be recognized that not only is it important that the mechanism perform functionally as expected, and provide the expected protections, but that the mechanism does not contribute to the risk of the LAN through a conflict with some other mechanism or functionality.

Each mechanism should be first tested independently of other services or mechanisms to ensure that it performs as correctly and provides the expected protection. In some cases, this may not be relevant to do, the mechanism may by design only interwork with other mechanisms. After testing the mechanism independently, the mechanism should then be tested in conjunction with other services and mechanisms to ensure that it does not disrupt the normal functioning of those existing services and mechanisms. The implementation plan should account for all tests, and should reflect any problems or special conditions as a result of the testing.

After all mechanisms are implemented, tested and are found acceptable, the list of priorities for LAN security should be reexamined. The risk associated with the threat/vulnerability relationships should now be reduced to an acceptable level or eliminated. If this is not the case, then the decisions made in the previous steps should be reconsidered to determine what the proper protections should be.

References

[1] Chang, Shu-jen, *Priorities for An Agency Local Area Network Security*, October, 1992.

[2] *Proceedings: National Computer Security Conference 1986*, pp.62-70.

[3] Barkley, John F., and K. Olsen, *Introduction to Heterogenous Computing Environments*, NIST Special Publication 500-176, November, 1989.

[4] *A Guide to Understanding Discretionary Access Control in Trusted Systems*, NCSC-TG-003, Version 1, September 30, 1987

[5] Smid, Miles, E. Barker, D. Balenson, and M. Haykin, *Message Authentication Code (MAC) Validation System: Requirements and Procedures*, NIST Special Publication 500-156, May, 1988.

MANDATORY POLICY ISSUES OF HIGH ASSURANCE COMPOSITE SYSTEMS

Jonathan Fellows
Grumman Data Systems
4015 Hancock Street
San Diego CA 92110

ABSTRACT

It is a commonly heard opinion that high assurance secure distributed systems - those that have a TCSEC level of B3 or A1 - are beyond the current state of the art. This paper argues that this need not be the case - that a combination of existing implementation and assurance techniques can meet high certification requirements for distributed systems. This paper's viewpoint is that the current lack of such systems is not the result of an inadequate technology base, but is due more to market forces and lack of interface standardization.

Distributed enforcement of a mandatory security policy in a multiple client/server architecture is analyzed, with an emphasis on the parallels with the mechanisms and assurances of a classical stand-alone Trusted Computing Bases (TCBs). Policies that support a state machine mandatory policy model are reviewed, and the impact of the distribution of these policies on implementation mechanisms and assurance approaches is explored.

1. INTRODUCTION

A multilevel secure distributed system must satisfy a mandatory system level security policy that confines information flows on the basis of security labels. It is rarely possible to develop such a system completely from scratch - a more realistic scenario is that individual components have been independently developed to the same standard of assurance. We assume that no single component controls policy definition or enforcement for the system as a whole.

The policies addressed in this paper are limited to the label-based Mandatory Access Control Policy described in the U.S. Department of Defense Trusted Computer Security Evaluation Criteria (TCSEC) [DOD85], as well as a number of supporting policies needed to support the mandatory policy. This limitation in scope is not meant to ignore the importance of other policies and security services to composite systems, such as discretionary access control, data integrity, assured service, and authentication. Mandatory policy enforcement was chosen as the topic for this paper in the belief that it is a simpler matter than those just mentioned, and that demonstration of composite trusted systems should proceed first with simple policies.

The most common paradigms for connecting distributed components are message passing, where an output of one component becomes an input of another, and remote operations, where a subject in one component invokes an operation on an object managed by a remote component. Both conventions are capable of serving as the basis for a distributed model of secure computation. We chose to examine distributed system composition from the remote operations point of view because the resulting architecture remains similar to well known worked examples of stand-alone reference model architectures. Our hope is that the precedents that have been established for stand-alone systems will apply to remote server systems as well.

The remote operation paradigm is at the heart of the increasingly popular Client/Server architecture, where a set of resources are managed by a server which may have a dedicated hardware platform. The resources are accessed remotely using a Remote Procedure Call (RPC) protocol. Secure versions of the RPC mechanism are the heart of the approach advocated in this paper. We believe that this mechanism will lead to composite systems that can be assured at the A1 level with only moderately more effort than equivalent stand-alone systems.

There is considerable ongoing work in the area of secured RPC protocols. The Kerberos system offers an identification and authentication service for client/server architectures, but does not address mandatory policy issues. The Trusted Systems Interoperability Group is in the process of defining a trusted Network File System commercial standard, but has not yet published results. Various OSI groups are investigating secure remote operations, but none have advanced beyond draft status.

2. SUPPORTING POLICIES

Access control models such as Bell and La Padula [BLP76] provide a useful high level model of mandatory policy enforcement, but they do not suffice in themselves to characterize the objective of information flow confinement[1]. A number of supporting policies are needed to approximate this confinement objective. The assurance of these supporting policies has often not been supported by formal analysis, even at the A1 level. This section reviews these supporting policies, and explores the issues of implementing them in a distributed system.

2.1 Entelechy

This obscure term, which is derived from a Greek word denoting "proper usage", was introduced by Kelem in [Fell87] to describe checks that are made to assure that individual "read" or "write" operations are properly constrained. The Bell and La Padula model is based on checks made at the time access is granted to an object, with the assumption that individual read and write operations are mediated by a hardware-based mechanism[2], which is not explicitly modeled.

There are two reasons to question the generality of this convention: (1) some interpretations of the access control model apply to objects whose read and write operations are implemented entirely by software mechanisms, such as file or database servers, and (2) some objects are accessed by a *stateless* service model, where each read and write is an independent event that does not depend on any prior service events. In the first case, explicit modeling of software-based checks that reads and writes are confined to a previously mediated context seems a necessary component of assurance. In the second case, forcing a single stateless service event to be modeled as an atomic sequence of three model events (e.g. request, read, release) is unnecessarily awkward, and calls into question the fundamental status of current access in the model.

Entelechy policies either constrain "reads" and "writes" within a stateful context that has previously been mediated for subject/object label consistency, or they require mediation of each stateless "read" and "write" individually. Examples of stateful services include memory segments, file systems, database systems, and connection oriented messaging. Examples of stateless services include some distributed file

1. This was not necessarily the intent of the original authors.

2. The original models were similar to the Multics operating system, where segmentation hardware performed these checks for memory segment objects.

systems and connectionless messaging. As can be seen from the duplications in these lists of examples, the choice of stateful or stateless service is not always inherent in the choice of object to be managed. Sometimes a design decision for a given access control system may depend more on the statefulness of the enforcement mechanism than on the service offered.

2.2 Model Data Stability

The simple security and *- properties constrain only current access. Any realistic model of secure systems must also provide for controlled changes in value of the remaining elements of the model. This simply means that, most of the time, we expect the population of subjects, objects, and their labels to be stable from one state of the model to the next. When these elements of the model do change, as when subjects or objects are created or deleted, we expect that role-based controls will be invoked which limit these actions to appropriate subjects. This is the heart of a number of criticisms of the Bell and La Padula model [McLe87].

Label stability is particularly sensitive. We often expect the label lattice to be stable over the lifetime of the system, and for label function values to be stable for the lifetime of the labeled subject or object (for objects, this is the "tranquility" principle). These expectations may be difficult to fulfill in federated systems, as is discussed later in this paper.

2.3 Model Data Integrity

Because the access control logic for a standalone system is maintained in a dedicated domain and is protected from tampering, the system can establish a user's identity at logon and reliably associate the user's logon session level with all subjects created to act on behalf of that user. All of the information needed to mediate access is available in one place.

When mediating access requests that involve more than one component system, it may be necessary to communicate label values from where they are maintained to where the access check is made. The communication channel that carries this information must preserve the label value and its associations unchanged. This requirement is similar to the trusted path requirement for authentication of users, except that here we are dealing with mutual authentication of trusted components.

2.4 Covert Channels in Operations

Covert channels can sometimes be viewed as unmodeled information flows. A state machine model of access control relies on characterization of operations on objects as read and/or write operations, depending on the direction of flow of information between subject and object. In order to attain a desired level of abstraction, this characterization often ignores obscure reverse-direction information flows in operations that are modeled as read-only or write-only. This need not be harmful, but rather can be viewed as a way to control the amount of complexity that is dealt with within the model, and as a way to factor assurance efforts. Well known sources of covert channels include object existence in a shared name space, resource locks on shared objects, message length encoding, and flow control of message based interfaces.

The bottom line is that we choose to allow covert reverse information flows in operations, while modeling them as one-way. As long as the mechanisms that implement read and write operations are under the complete control of the trusted access control mechanism, covert information flows can often be identified and limited in bandwidth.

3.COMPONENT LEVEL MODELS

The mandatory policy model adopted for this paper is the state machine of the Bell and La Padula model. The choice of this model for a distributed application is controversial within the security research community, but we believe that it has three major virtues: (1) it fits well with the popular client/server model of system distribution; (2) it has established precedents as the basis for a TCSEC A1 level evaluation, and (3) its focus on internal state provides implementation design guidance.

Rather than view a distributed system as a single state machine, we will view a distributed system as a family of state machines that may be connected to each other by sharing elements. The following notation will be used to refer to the elements of the security models for each component, C[i]:

L[i]: Label lattice of C[i], defines a set of labels and a label comparison operator

S[i]: Subjects managed by C[i], assumed to be processes executing on a single machine.

clearance[i]: Subject clearance attribute function for C[i], S[i] -> L[i]

O[i]: Objects managed by C[i]

label[i]: Object label attribute function for C[i], O[i] -> L[i]

A[i]: Set of access operations (modes) allowed by C[i]

CA[i]:Current Access matrix for C[i], S[i] x O[i] -> Powerset(A[i])

Each system enforces a security policy with simple security and *- properties that state limitations on the values that the current access matrix can take. The effect of these properties are the familiar "no read up" and "no write down" restrictions.

This notation defines a class of models that provides a state machine description of the access controls necessary to confine operations on labeled objects so that information cannot flow from a "higher" labeled object to a "lower" labeled object. Demonstrations of this *confinement property* follow an inductive method: first show that the initial state is secure, then show that all transitions from a secure state result in a secure state.

4. COMPOSING SECURE SYSTEMS

Our way of looking at trusted client/server systems is as a collection of individually secure state machines which share some of their elements. In principle, any of the elements in the state machine model could be shared, but limiting the amount of coupling results in a more modular architecture. The following sections demonstrate one such scheme of sharing individual model elements.

4.1 Shared Labels

Coupling of the label lattices of component systems is a prerequisite for system composition. The simplest case of label coupling is where all systems to be composed share the same label lattice, so that L[i] = L[j] for all i and j. All components implement the same label comparison operation, so that label ordering is the same on all components. This case is frequently too simplistic to handle the way labels are used in real applications. The following situations have had to be accommodated in real systems:

- Different systems frequently implement private label data types. In this case a label translation must be performed for every labeled interaction between systems. The best way to handle this is with a single system level label standard, with translations performed on import and export. The label translation function must be order preserving.

- Different systems can operate over different accreditation ranges, so that each system recognizes some sublattice of the overall system lattice. This may result in a requirement for label coercion, or relabeling, at the time of a trusted import or export.
- Different systems may represent subject clearances with different data structures. Real life examples include labeling subjects with a single label that represents the subjects highest clearance level, a range of labels that defines the sublattice over which a subject can operate, and a list of individual labels that a subject is allowed.
- An individual system may extend the system label lattice with local values used for local TCB structuring purposes.
- The propagation of a system-wide change in labeling to the individual L[i] requires global consistency assertions that are difficult to assure in practice. The need for system level label changes is real, since a compartment may have a lifetime less than that of the system.

4.2 Shared Subjects

Sharing subjects means that a local subject of one component state machine can remotely invoke a mediated operation on an object under the control of another state machine. Each state machine mediates requests from both local subjects, for which it maintains subject clearances, and remote subjects, for which it does not. This means, in effect, that each state machine's subject label function, clearance[i], is distributed across all of the state machines within a given system. It is important to remember that subjects are not users, but processes executing on behalf of users. A single subject is executing on a single machine, and no entry in the label function is under the control of more than one state machine.

A unitary Trusted Computing Base (TCB) has the advantage that the definition of subjects and their security relevant attributes is completely under the control of the TCB. Since the TCB usually controls the process abstraction of the system, the operation invocation mechanism at the TCB boundary provides a reliable means of identifying the subject performing a mediated operation. Since the internal TCB storage that contains the subject clearance information is assumed to be high integrity, when the TCB retrieves a subject's clearance, it believes the value it finds.

Mediation of remote operations requires high integrity equivalents to these properties. Before mediation of a remote operation can proceed, the clearance of the remote subject must be established in a way that cannot be spoofed or counterfeited. A TCB to TCB communication channel must be established using a mechanism that is as strong as the hardware domain mechanism of the unitary TCB.

The problem of reliably communicating a remote subject's clearance, which is the key issue in supporting distributed mandatory policy enforcement, can be solved in a number of ways, some of which will be discussed in a later section. This is actually a significantly easier problem than reliably (and unambiguously) establishing a remote subject's associated user identity, which is needed to support discretionary policies and audit accountability.

4.3 Private Objects

In order to preserve the desired analogy between stand-alone and distributed systems, we adopt the convention that each trusted state machine in the system manages its own set of objects, which are disjoint from the objects of any other machine. This means that all objects in the composed system remain completely under the control of a single component level TCB. Considerably more general relationships are possible between objects of components to be composed, such as hierarchical containment or joint management. These more

general schemes result in distribution of a state machine's O[i], label[i], and CA[i] elements, making it significantly more difficult to analyze the system for high assurance TCSEC levels.

When all operations on the objects managed by a particular machine are mediated by the TCB of that machine, the reference monitor protection mechanisms and the policy specifications of a particular machine are nearly identical to those of a stand-alone TCB. In particular, the representation of objects, object labels, and current access sets are not impacted by the remote operation distribution mechanism. The representation of these entities usually takes the majority of the effort in developing formal specifications and showing correspondence to an implementation.

4.4 Remote Operations

A *remote procedure call* (RPC) [Nels87], is an application level protocol which implements a remote operation on top of a message-based communications service. This section discusses some of the issues involved in securing an RPC service. The beauty of the trusted RPC approach is that it can implement one-way operations (i.e. read-down or write-up) on top of two-way communications. By way of contrast, message passing schemes are usually limited to two-way same-level peer communications,[3] leading to a need for trusted subjects in many applications.

Remote procedure protocol implementations, such as the one defined in [RFC88], typically have the following properties:

- Input parameters of the procedure call are marshalled into one or more request messages from a client to a server.
- Output parameters of the procedure call are marshalled into one or more response messages from a server to the requesting client.
- Multiple concurrent calls are supported, so that an implementation needs to be able to associate requests and responses from the same call, which may share a unique transaction identifier.
- Optional authentication information may be carried in request and response headers.
- The version number of the RPC protocol is included in request and reply messages.
- No data integrity mechanisms are added to those of the underlying transport protocol.

To these functional properties, a multilevel version of RPC must add the facility for reliably communicating the clearance of the requesting subject, and for binding this clearance to a request in a way that cannot be tampered with.

We assume that each RPC protocol for a mediated operation is part of the trusted computing base of both the client and the server component. The fact that the server is part of the TCB, and trusted to send and receive messages of different labels is what allows the implementation of one-way operations, even if the transport mechanism enforces its own label-based policy.

A classical unitary TCB uses hardware mechanisms to guarantee that a mediated operation results in information flow only between the subject and object identified in the operation. A secure RPC mechanism must likewise guarantee that the messages it employs are only visible to the client and server TCBs, and that results are communicated only to the originating subject. This will require high assurance of transaction identifier data integrity and binding of transaction identifiers to messages.

3. The Boeing Secure LAN [Schn87] is an exception to this rule, with its support for one-way TCP connections.

5. IMPLEMENTATION ISSUES

This section briefly describes some of the implementation options for secure RPC, focusing primarily on mechanisms that provide the supporting policies introduced in Section 2.

5.1 Entelechy

Entelechy deals with confinement of individual read and write operations to a mediated context. For a stateless service, the client component must provide the clearance level associated with the calling subject to the server component every time an operation is invoked. For a stateful service interface, the RPC operation that establishes a service session can provide the subject clearance to the server component. Subsequent RPC calls within the context of this established session must establish their association with the session, Since we are dealing in this paper only with mandatory policy, we do not address the authentication of user identity on a per-call or per-session basis.

Stateless services, which were chosen for the Network File System [RFC89] have the advantage of not having to reliably maintain state information in the presence of component failures. Similar benefits accrue to a secure stateless service, where each request is mediated afresh, obviating the need to deal with the recovery problems of half-stateful associations (e.g. a client with a secure file handle but no server context).

5.2 Model Data Stability

Ultimately, even mandatory policies are identity-based, in that the clearance associated with a subject when it is created is the clearance of the user on whose behalf the subject is executing. Maintaining user clearances independently on each component machine, while perhaps meeting the letter of high assurance requirements, is awkward and error-prone.A trusted directory service that provided a high integrity central point of definition for user clearances would be a significant improvement. Such a service is described in [Linn90].

The set of operations offered by a service are defined by the RPC protocol definition for that service. If this changes over time, the effect of different components executing different versions of the protocol could cause security violations. This is a model data integrity issue that leads to a need to treat RPC version numbers as a high integrity element of the protocol. The version number should change any time a change is made to operations or labels.

5.3 Model Data Integrity

The communication security service requirements of multilevel secure RPC are:

- Identification with high data integrity of the subject clearance associated with a call.
- Binding of subject clearance to an RPC request.
- Correct association of result data with an original request (also an entelechy issue).
- Delivery of call results only to the requesting server (also an entelechy issue).
- Confidentiality of RPC data in transit between client and server TCBs.

The requirement to securely communicate subject clearances, transaction identifiers, and RPC version numbers between client and server TCBs is not currently supported by TCP/IP protocols. BLACKER supports appropriate IP level confidentiality, but does not provide the needed integrity services. [Fell87] describes application specific use of embedded encryption to provide similar integrity services for the administrative control functions of a secure network, using message authentication checksums. Similar techniques could be used to directly provide integrity of RPC messages.

Appropriate integrity services are available from both the SP3 and SP4 protocols, which are SDNS variants of the OSI CLNP and TP4 protocols. A secure RPC could specify data integrity service for both request and reply messages. SP 3 is probably the better choice, since typical connection-based cryptographic mechanisms, such as that used for SDNS SP4, currently require several seconds to establish a connection - clearly too much overhead to wrap around each RPC call.

5.4 Covert Channels

Each of the component state machines, as part of its certification, must have been subject to covert channel analysis of the trusted service it provides. Composing different state machines with a trusted RPC protocol introduces two new opportunities for covert channels (1) information flows introduced by the RPC protocol itself, and (2) composition of covert channels from different components.

The RPC protocol itself does not present a TCB interface directly to untrusted subjects, which see only the procedural interface defined by a server component. The RPC implementations are a part of both the client and server TCBs, which means that untrusted subjects never have the ability to sense protocol control information such as header fields, message lengths, or address information. Of course, some control information ultimately is returned to subjects in the form of operation result status, but this was already the case for the undistributed form of the server TCB.

The presence of multiple trusted servers in a system does introduce the possibility of cascading covert channels end to end, so that multiple TCBs are involved in a covert signalling path. However, if each single TCB covert channel meets TCSEC bandwidth guidelines for its assurance level, then the cascaded channel will meet the guideline as well.

6. ASSURANCE ISSUES

Our chosen convention for sharing model elements of component state machines assures that most model elements remain under the control of a single TCB. The effect of this convention is that the mandatory security policies, reference monitor mechanisms, and assurance evidence of a stand-alone TCB is not significantly altered when the TCB becomes a server in a client/server architecture. The exception is the subject clearance function, which is distributed over all of the components of such a system. Even here, our convention is that clearance entries are partitioned in the system - i.e. no entry is maintained by more than one TCB. So again, previously existing component controls over individual subject clearance entries should meet high assurance requirements.

Composition through remote operations requires distributed assurance primarily for the supporting policies mentioned in Section 2. At the A1 level of assurance, there is precedent for the use of rigorous engineering analysis, as opposed to formal modeling, of these policies. In terms of the size and complexity of the TCB to be assured, the remote procedure call implementation is considerably less complicated than a reliable connection oriented protocol like TCP or TP4. This is due to the decision taken in [RFC88] not to address reliability issues in the RPC protocol.

Clients and servers of a given system need not share the same underlying TCB architectures. This fact can serve to strengthen system level assurance through the use of servers which execute on local TCBs whose architectural tradeoffs have been decided in favor of TCSEC assurance requirements. Components which support general purpose clients are driven by market forces to provide an environment that is as close as possible to an already accepted commercial interface. Specialized secure servers are not driven by this requirement, and yet

can still support a standard interface such as NFS with an RPC service.

7. REFERENCES

[BLP76] Secure Computer Systems: Unified Exposition and Multics Interpretation, MTR-2997 Rev.1, MITRE Corp., Bedford, Mass., March 1976.

[DIA86] DODIIS Network Security Architecture and DODIIS Network Security for Information Exchange (DNSIX), Defense Intelligence Agency, May 5, 1986.

[DOD85] DOD 5200.28-STD, Department of Defense Trusted Computer Evaluation Criteria, National Computer Security Center, December 1985.

[DOD87] NCSC-TG-005, Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, National Computer Security Center, July 1987.

[Fell87] Fellows, J., Hemenway, J. Kelem, N. and Romero, S., "The Architecture of a Distributed Trusted Computing Base," Proc. 10th National Computer Security Conference, 1987.

[Linn90] J. Linn, "Practical Authentication for Distributed Systems," Proc. IEEE Symposium on Security and Privacy, 1990.

[McLe87] J. McLean, "Reasoning About Security Models," Proc. 1987 IEEE Symposium on Security and Privacy, April 1987, pp 123 -131.

[Nels81] B. Nelson, "Remote Procedure Call," CSL-81-9, Xerox Palo Alto Research Center, May, 1981.

[Nels87] R. Nelson, "SDNS Services and Architecture," Proceedings of the 10th National Computer Security Conference, September, 1987, pp. 153-157.

[RFC88] Sun Microsystems, "RPC; Remote Procedure Call Protocol Specification, Version 2," Internet RFC 1057, June, 1988.

[RFC89] B. Nowicki, "NFS: Network File System Protocol Specification," Internet RFC 1094, March 1989.

[Schn87] D. Schnackenberg, "Applying the Orange Book to an MLS LAN," Proc. 10th National Computer Security Conference, 1987.

MEDIATION AND SEPARATION IN CONTEMPORARY INFORMATION TECHNOLOGY SYSTEMS

Marshall D. Abrams, Jody E. Heaney, Michael V. Joyce

The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102

ABSTRACT

This paper reexamines the concepts introduced in the Anderson Report and the interpretations of those concepts in the *Trusted Computer System Evaluation Criteria*, *Information Technology Security Evaluation Criteria* , and the *International Organisation for Standardisation (ISO) Access Control Framework*. The authors contend that there has been an evolution in the understanding of these concepts and features and that the evolution is useful. The authors further suggest that the fundamental feature of separation is necessary to trust technology and that this concept has been overlooked in the evolutionary process.[1] The authors suggest that the use of separation mechanisms and providing support for multiple policies are issues that need to be incorporated into the evolutionary cycle as more mature interpretations of the Anderson Report concepts are formulated.

1. INTRODUCTION

Contemporary Information Technology (IT) systems are required to satisfy many different needs. These needs span a broad spectrum and include characterizations such as confidentiality, integrity, availability, safety, and criticality. Most prior work in the development of trusted technology has concentrated on the properties of an IT system that satisfy only the confidentiality requirement. In this paper, the original definitional statements of the basic concepts are reviewed, and the evolution of the concepts and their use in more recent standards are considered. A major consideration, the use of multiple policies, overlooked in all existing standards is discussed. The impact of multiple policies on the existing concepts is analyzed. It is suggested in section 3 that both the reference monitor (RM) and Trusted Computing Base (TCB) concepts may require further evolution to address the broader range of properties required of trusted IT systems prior to the development of future standards. We present the view that separation kernels, as fundamental constructs, may provide a potential solution for the further development of trusted IT systems which focus on the use of multiple policies.

2. HISTORICAL PERSPECTIVE

In this section, a historical perspective is provided via a brief review of the original reference monitor and TCB concepts. This review is followed by discussion of the more recent *Information Technology Security Evaluation Criteria* (ITSEC) [1] and the International Organization for Standardization (ISO) *Access Control Framework* [2].

2.1 THE ANDERSON REPORT

The Anderson Report [3] introduced several important concepts that continue to be fundamental in the design and architecture of technical protection mechanisms against unwanted utilization or modification of IT

1 This work was funded by The MITRE Corporation and the Department of Defense, Number DAAB07-91-C-N751, Numbers 8812Q and 96440.

resources. The terms "secure" and "trusted" have been applied to IT products and systems that provide such protection. While "trusted" refers to the protection provided by IT, "secure" also includes administrative, procedural, and physical security.[2] The technology that supports such trust is more recently referred to as "trust technology."

The Anderson Report based its conclusions and recommendations on studies of 1970s systems. Remarkably, many of the conclusions remain valid twenty years later. In fact, the authors find some of the conclusions and recommendations in the Anderson Report more generally applicable today than their interpretations in subsequent documents. Nevertheless, it is valuable to review the conclusions and recommendations and to suggest updates and extensions for the 1990s.

The Anderson Report identified the following three requirements to defend against the malicious user threat: (1) an adequate system access control mechanism; (2) an authorization mechanism; and (3) controlled execution of user programs and operating system service functions. Taken together, the three requirements express the need for controlled sharing.

The reference monitor concept was introduced as an ideal design to achieve controlled sharing. "The function of the reference monitor is to validate all references (to programs, data, peripherals, etc.) made by programs in execution against those authorized for the subject (user, etc.). The reference monitor not only is responsible to assure that the references are authorized to shared resource objects but also to assure that the reference is the right kind (e.g., read, or read and write, etc.)."[3] The reference monitor validates all access requests made by subjects for objects according to the access authority of the user. The relationship of these components is illustrated in figure 1.

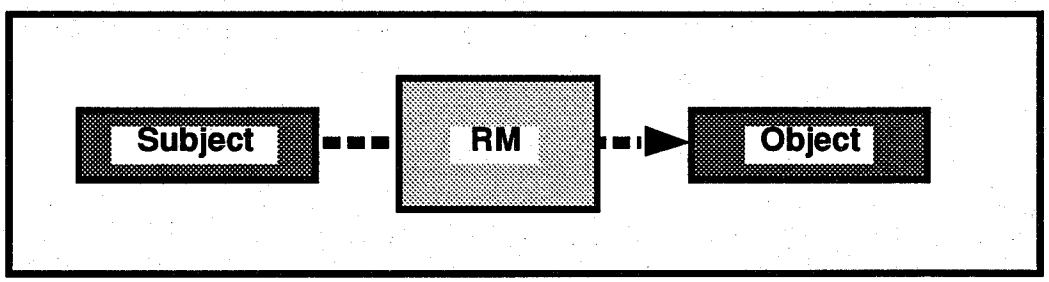


Figure 1. The Reference Monitor Concept

An implementation of an RM is called a reference validation mechanism (RVM). An RVM is the combination of hardware, software, and firmware that implements the RM concept. The Anderson Report adds the following guiding principles for the RVM:

- The RVM must be tamperproof.
- The RVM must always be invoked.
- The RVM must be small enough to be subjected to analysis and tests to assure that it is correct.

The Anderson Report goes on to "...develop the design for the security portion of a system,..." which it calls the (security) Kernel.[3]

---

2 This distinction is attributed to Stephen Walker [4]. A second differentiation views the state of being *secure* as an ultimate objective but recognizes systems that are good enough to achieve *required* security objectives in a specific environment by stating that these systems are *trusted*. Trust implies a value judgment. While it is practically impossible to achieve a (completely) secure system, it is possible to achieve a trusted system.

3 Note that the Anderson Report referred to the design variously as a Kernel or security Kernel. In quotations from the report, the "k" in kernel is capitalized per the original material, but in all other instances the "k" is lower case. In material not related to the Anderson Report, we use the colloquial terminology, "security kernel".

2.2 TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA

By 1983, when the RM, RVM, and kernel concepts were incorporated in the original version of the *Trusted Computer System Evaluation Criteria* (TCSEC) [5], the terminology had evolved so that concept and implementation were referred to as "TCB" and "security kernel," respectively. The Anderson Report states "...the security Kernel design incorporates the reference validation mechanism, access control (to the system), and authorization mechanisms. Further, it will probably incorporate the administrative programs to represent and maintain user and program authorizations. ... The requirement for controlled execution of a user's program (or a program being executed on his behalf) is merely a statement that requires the references made by the program to be authorized for the user on whose behalf the program is being executed." [3] The TCSEC provides a definition for TCB. The TCB is: "The totality of protection mechanisms within a computer system — including hardware, firmware, and software — the combination of which is responsible for enforcing a security policy. A TCB consists of one or more components that together enforce a unified security policy over a product or system. The ability of the TCB to correctly enforce a security policy depends solely on the mechanisms within the TCB and the correct input by system administrative personnel of parameters (e.g., a user's clearance) related to the security policy." [5]

There are advantages in the TCSEC definition of the TCB. It is convenient to have a concise term to refer to all the security-relevant functionality. It is useful to define the security perimeter for enforcing a security policy. One of the primary advantages is that the TCB defines the boundary of an evaluated product. From the evaluation perspective, the vendor is responsible only for what is within the TCB; similarly, the evaluators need only consider what is included in the TCB. When used this way, everything included in the TCB is "trusted"; everything outside the TCB is "untrusted." Thus the TCB definition establishes a boundary separating trusted from untrusted code. The ultimate logical extension of this concept is that the TCB should be sufficiently strong to prevent untrusted hostile code, even code written by an adversary, from violating security policy.

There are also disadvantages in the TCSEC definition of the TCB. First, no framework was provided to indicate what belonged in the trusted portion of the code. Thus, the trusted part of the system often grew very large in proportion to the untrusted code. Second, it fails to differentiate regarding the quality of the implementation's structure. All TCSEC evaluation classes have TCBs, but it is only at class B2 that requirements are introduced to identify the TCB modules that contain the RVM and to explain why it is tamper resistant, cannot be bypassed, and is correctly implemented. Third, the TCSEC prescribes a specific security policy that the system is required to enforce. That is, the TCSEC essentially defined an explicit policy for confidentiality. The level of detail in policy stipulation means that the TCSEC fails to address situations in which more than one security policy is required to be enforced. One possible representation of such complexity might be a TCB for each policy; the relationship among these TCBs could be arbitrarily complex.

2.3 ITSEC

In 1991, the European Communities published the *ITSEC: Information Technology Security Evaluation Criteria* [1]. The ITSEC builds on the TCSEC and other national IT security criteria. It is instructive to consider some of the terminology used in the ITSEC and to contrast it with earlier usage.

The ITSEC emphasizes a distinction between an IT system (a specific IT installation with a particular purpose and known operational environment) and an IT product (a hardware and/or software package that can be bought off the shelf and incorporated in a variety of systems). The ITSEC uses the term "Target of Evaluation (TOE)" to refer to both, believing that "...it is important for the sake of consistency that the same security criteria are used for both products and systems. ... A TOE can be constructed from several components. Some components will not contribute to satisfying the security attributes of the TOE. Other components will contribute to satisfying the security objectives; these components are called security enforcing. Finally, there may be some components that are not security enforcing but must nonetheless operate correctly for the TOE to enforce security; these are called security relevant." [1]

The ITSEC essentially differentiated between the parts of a "TCB," identifying them as security enforcing or security relevant. It is not clear how actively or directly a component must contribute to satisfying security objectives to be considered security enforcing as compared to security relevant. It is clear that the RVM is security enforcing. Since identification and authentication (I&A) is a prerequisite for the RVM to enforce a

security policy, we consider it to be security enforcing. Audit also contributes to enforcing security policy but is not a prerequisite for the RVM; therefore, we consider audit to be security relevant.

The ITSEC did not include a specific confidentiality policy, although it does strive to establish a logical link to the TCSEC. "Throughout the TCSEC, the combination of both the security enforcing components and the security relevant components of a TOE is often referred to as a Trusted Computing Base (TCB). TCSEC TOEs representative of the higher classes in division B and division A derive additional confidence from increasingly rigorous architectural and design requirements placed on the TCB by the TCSEC criteria. TCSEC classes B2 and higher require that access control is implemented by a reference validation mechanism. ... For compatibility with the TCSEC, the ITSEC example functionality classes F-B2 and F-B3 mandate that access control is implemented through use of such a mechanism." [1]

The ITSEC also asserts equivalency with the TCSEC TCB requirements. "At higher evaluation levels the ITSEC places architectural and design constraints on the implementation of all the security enforcing functions. Combined with the ITSEC effectiveness requirements that security functionality is suitable and mutually supportive, this means that a TOE capable of meeting the higher ITSEC evaluation levels and which provides functionality matching these TCSEC-equivalent functionality classes, must necessarily satisfy the TCSEC requirements for a TCB and use of the reference monitor concept." [1] One hundred percent compatibility remains elusive [6].

2.4 THE ISO ACCESS CONTROL FRAMEWORK

The ISO Access Control Framework [2] draws upon many of the concepts just reviewed. The framework identifies component functions within the access control decision making process (see figure 2). One function, the Access Control Decision Function (ADF), decides whether an initiator may perform an action on a target.[4] The value returned by the ADF represents a decision. A second function, the Access Control Enforcement Function (AEF), receives the decision and enforces it.

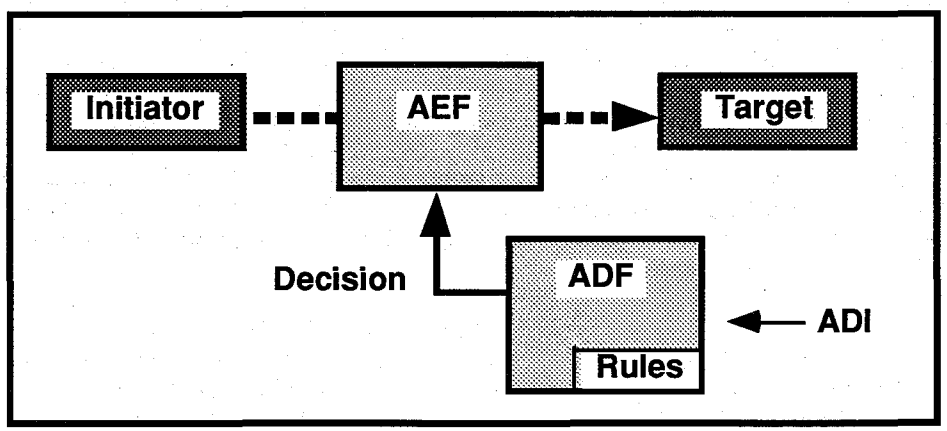


Figure 2. ISO Access Control Framework

The ISO Access Control Framework categorizes the information needed to make an access control decision. Two general categories of information are identified: access control rules and Access Control Decision Information (ADI). The rules, logical rule expressions instantiating policies, establish the constraints on actions performed by the initiator. The ADI is that information made available to an ADF about the initiator, target, and action, including contextual information that the ADF needs to make its access control decisions.

4 "Initiator" and "target" are the terms ISO uses for active and passive entities in a system.

2.5 RECAPITULATION

Briefly reviewing, the Anderson Report started a discussion of concepts related to the development of trust technology. That discussion appeared at a high level of abstraction and has weathered the intervening years well. That is, the original concepts it presented are still valid today. The TCSEC introduced the TCB concept, which included the notion of dividing the trusted and untrusted code. The Achilles' heels of the TCSEC have proved to be the incorporation of a specific confidentiality policy and the lack of differentiated quality standards for TCB implementations. The ITSEC, viewing the TCSEC as a training ground, did not include a specific policy and it specifically addressed the quality issue. In addition, the ITSEC began to differentiate between the various portions or components of a TCB. Finally, the ISO Access Control Framework is focusing specifically on the access control mechanism and is differentiating between the various components of that mechanism.

3. CONTEMPORARY CONCEPTS RAISE QUESTIONS

The evolution in the interpretation of the concepts introduced in the Anderson Report has been shown in the discussions of the TCSEC, ITSEC, and ISO Framework. The evolution in the understanding of these concepts is useful. Periodically it is good, however, to reevaluate the conventional view and adjust the evolutionary direction. The fundamental concept of separation is necessary to trust technology, but this concept has been overlooked in the evolutionary process. Moreover, as the trust IT arena pushes forward, the reality of multiple policies being enforced by the same system is being recognized as a necessity by more and more developers.

This section uses the observations noted in the preceding paragraph as a point of departure. The section begins with a comment on a drift in the interpretations of security kernel and RVM as originally presented in the Anderson Report – an unfortunate by-product of evolution. The issues of separation and mediation are dealt with and the roles of these concepts in achieving the goals specified in the Anderson Report are discussed. Finally, an approach is offered for dealing with multiple policies and for meeting the original goals specified for the reference monitor concept.

3.1 DESIGN AND ARCHITECTURE DISTINCTIONS

In the Anderson Report, the reference monitor, reference validation mechanism, and security Kernel were proposed as *one useful design* to counter the malicious user threat. In the TCSEC, a TCB embodying this design became *the one and only architecture* that was acceptable for preserving confidentiality. That is, while the Anderson Report suggested a way to approach the problem, the TCSEC described the only solution that would be accepted as appropriate trust technology. The ITSEC equivocates on this point, fully agreeing with neither the Anderson Report nor the TCSEC.

In addition to this distinction regarding accepted designs, one is compelled to mention the drift in interpretation of security kernel and RVM. The original documents referred to the security kernel as the security portion of a system. The security kernel was very inclusive and encompassed the RVM as well as access control, authorization, and administrative mechanisms. In current usage, the terms "security kernel" and "RVM" are sometimes nearly inverted. That is, the security kernel is construed to be equal to the reference validation mechanism. In addition, the implementation of a TCB is sometimes described as being composed of a security kernel *and* trusted processes. Using the original definitions, one should more correctly describe the TCB implementation as a reference validation mechanism and trusted processes, *or* as a security kernel. We believe that this distinction between reference monitor and TCB provides a measure of insight and that suppression of this distinction is regrettable.

3.2 SEPARATION AND MEDIATION

In the documents reviewed in section 2, the terms "separation" and "mediation" were found to be used interchangeably. These are two inherently different mechanisms. The goal of separation is rigorous isolation to gain integrity or confidentiality or other properties such as TCB self-protection. Appropriate separation of resources prevents the accidental intermingling of information requiring different forms of protection, and separation of processes prevents the intentional transfer of information. Separation is a common mechanism used for the control of information flows within a computer system. On the other hand, the goal of mediation

is controlled access, which is generally enforced based on an access control policy. Such a policy is typically unique and dynamic.

We believe that separation is the more fundamental of the two mechanisms, and that mediation can be applied on top of the foundation offered by separation. Separation inherently provides support for diverse needs and is more generic. Separation can provide the structural integrity necessary to support appropriate access control policies. From this perspective, we are inclined to agree with Rushby's [7] view of separation kernels or domain separation as being the appropriate base for an IT system.

Rushby and Randell [8] argued *against* using a security kernel as the only mechanism for security in general-purpose systems. (Note that their focus was the development of a distributed secure system as opposed to a secure operating system [OS].) In their paper, separation and mediation are treated as distinct logical concerns, with separation the more basic principle. Rushby and Randell discussed four types of separation mechanisms: physically separate components, temporal separation or periods processing, cryptographic separation, and logical separation. In order for separate components to communicate, they were required to belong to the same security partition (i.e., a group of components that form a community with agreements in place for communication between them).

Rushby [7] later considered TCBs for embedded systems. In this work, he proposed that a trusted embedded computer system should be structured in three layers (see figure 3). The lowest layer was a domain separation mechanism (DSM). The DSM divided the system into a number of separate execution domains, or virtual machines, and provided controlled communication channels among the domains. The middle layer contained a set of resource managers (REMs) that controlled the system resources.[5] The highest layer was the set of applications. The domain separation provided by the DSM allowed untrusted programs to operate in the domains and not interfere with one another. Both the DSM and the resource managers contained instances of the reference validation mechanism for adjudicating inter-domain communications and access to resources, respectively.

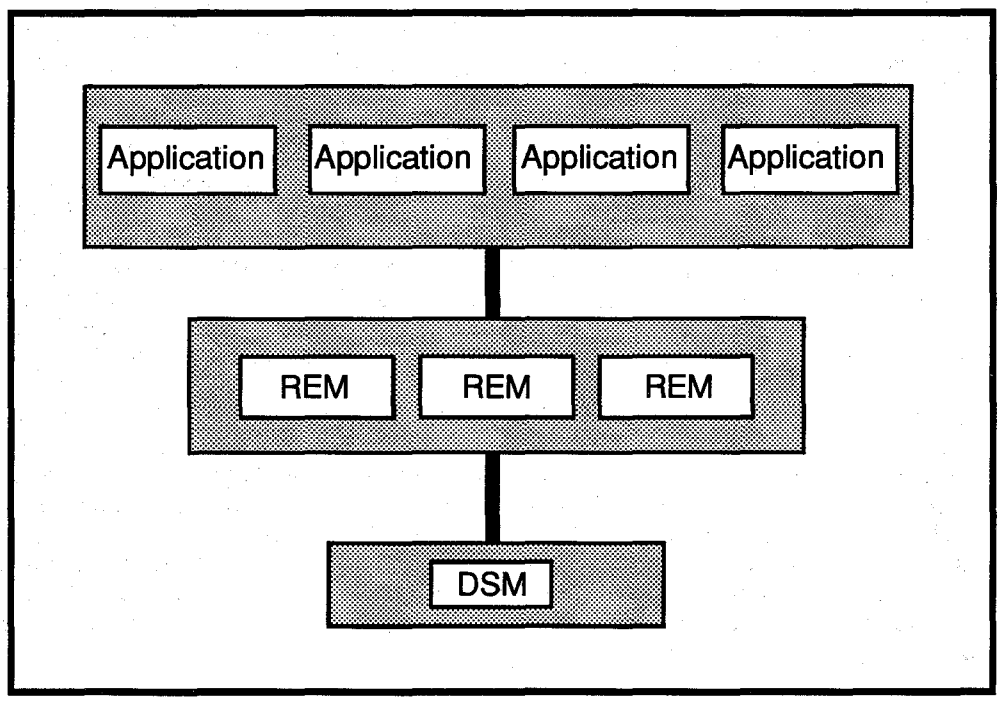


Figure 3. Domain Separation Kernel Concept

5 The REMs provide the abstractions of objects composed from the system resources; the file system manager is a typical REM.

Rushby [9] returned to the subject of kernels, though the orientation was safety kernels. Rushby asserted that "...kernelized system structures can provide rigorous guarantees that certain faults of commission will not occur." By this he meant that the kernel cannot enforce good behavior (positive properties); it can only prevent bad behavior (negative properties). Rushby asserts that kernel structures are well suited to guaranteeing negative properties (e.g., security) rather than positive properties (e.g., availability or safety). He based this view on the fact that a security kernel is located at the bottom of a hierarchical structure and, therefore, can enforce security outward without the cooperation of the rest of the system.

3.3 MULTIPLE POLICIES

The growing interest in new forms of IT system access control is challenging the conventional view of the access control process. The historical perspectives of access control, although they do not state the assumption, assume a single access control policy (ACP). Instead of an access control decision based upon a single ACP, the ACP enforced in future IT systems will likely be a composite of several constituent ACPs (e.g., ACP_1, ACP_2, ACP_3, ... ACP_n). This evolution is occurring without an adequate framework for identifying the presence of multiple policies and for understanding the interaction of these policies.

Support for the statement that future systems will contain multiple policies is diverse. The research community has been very active in examining access control policies beyond the Mandatory Access Control (MAC) and Discretionary Access Control (DAC) policies contained in the TCSEC [5].[6] The effect of some of this work has already been felt – the *Trusted Network Interpretation* (TNI) [19] includes integrity and availability evaluation requirements. Some research efforts [20, 21, and 22] have addressed the multiple policies issue directly.

A proposed extension to the ISO Access Control Framework [22] offers an alternative conceptual framework for dealing with the presence of multiple access control policies. There are three facets to this proposed framework (see figure 4). First, to address explicitly the presence of multiple policies in the IT system, individual Access Control Rule[7] sets are proposed, one for each access control policy. Next, instead of implementing multiple access control policies in a single Access Control Decision Function (ADF), multiple ADFs are proposed. This combination is intended to provide a clear, explicit, isolated context for each constituent policy.

The third facet of the proposed framework is a mechanism to combine the decisions of the individual ADFs. The participation of several ADFs in the adjudication of an access request results in several decisions. To make an access control decision, however, the decisions from the ADFs must be combined into a single access control decision to be acted upon by the AEF. To accommodate this situation, a Metapolicy Function (MPF) has been proposed. This new function is logically positioned between the AEF and the ADFs and combines the decisions of the various ADFs into a single vote passed to the AEF. (Note that in practice, it may be reasonable to incorporate the MPF within the AEF. We separate them here for clarity of concept and by function performed.) The way in which these decisions are combined is defined by a set of Metapolicy Rules (MPRs).

The proposed extension of the ISO Access Control Framework provides a modular view of the access control process with new functions to address the presence of multiple access control policies. The AEF, the Metapolicy Function, and the ADFs are the access enforcement components that work together to mediate each access by an initiator to a target. In terms of the Anderson Report, these components constitute a reference validation mechanism. The extended framework shown in figure 4 also identifies the rules and data (i.e., user attributes and policy) upon which the access enforcement components are dependent. These data provide a basis for the reference validation mechanism to make the final access control decision. To ensure correct, consistent operation of the reference validation mechanism, these data must be adequately protected and, therefore, are included in the TCB.

---

6 For a cross-section of related work, see references 10, 11, 12, 13, 14, 15, 16, 17, and 18.

7 A rule-based access control policy uses rules predicated on the relationships between the attributes of an initiator and a target. These rules usually rely on a comparison of the sensitivity of the resources being accessed and the possession of corresponding attributes of users, a group of users, or entities acting on behalf of users [23].

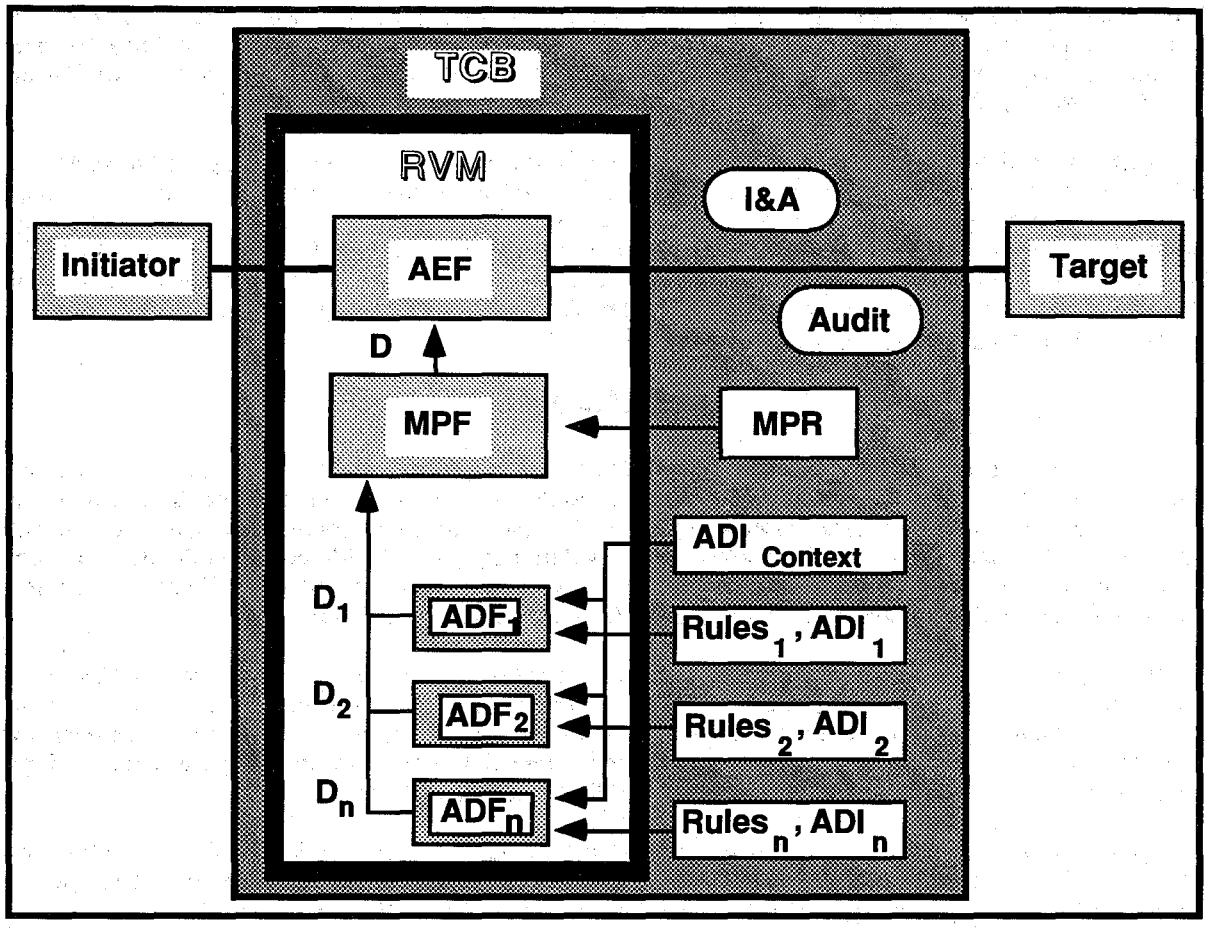


Figure 4. Extended ISO Access Control Framework

Separating the enforcement components and support data in the framework extensions may aid in achieving the goals established by the Anderson Report. By placing the policy definition components (i.e., Access Control Rules and Metapolicy Rules) outside the reference validation mechanism (but still within the TCB boundary), the reference validation mechanism is smaller, easier to understand, and more resilient. Isolation of the support data should allow more straightforward implementations of policy changes. A simple policy change could involve merely updating the Metapolicy Rules or Access Control Rules. A drastic change could involve a new set of Metapolicy Rules and the addition of a new ADF and Access Control Rules combination. But in all cases, a change is well defined and localized.

3.4 POLICY-ENFORCING APPLICATIONS

The view that the OS is the *only* enforcer of an access control policy is also rapidly changing as the complexity of the applications needed to support the operation of a contemporary enterprise increases. Instead of just the access control policies being enforced by the OS, many applications create objects and enforce their own access control policies. We refer to such applications (e.g., database management systems) as "policy-enforcing applications (PEAs)."

PEAs are related to the TCB via their policy-enforcing role. Drawing upon the ISO Access Control Framework concepts, a PEA contains data to define the access control policy and a function (i.e., ADF) to enforce the policy. PEAs may also include their own resource managers. Using this view, we determined that while an application's ADF is logically part of the application, it is also logically part of the TCB.

(Obviously, not all applications are PEAs and therefore not all applications will have an ADF.) The relationship between the non-security-relevant part of a PEA, the policy-enforcing part, and any resource management need not be one-to-one. We observe that a TCB should include part of the PEAs as well as the security enforcing and security relevant parts of the OS.

As more functions are added, it is easy to see that the TCB is quickly becoming too large and unwieldy for high-assurance modeling and security analysis when multiple policies, including those enforced by applications, are present. Alternative architectures are needed so that analysis of a baroque TCB overloaded with policy-enforcing functions can be avoided. We suggest that a domain separation mechanism may prove to be the fundamental mechanism in the TCB on which the security of the entire system can depend. A domain separation mechanism, such as that suggested by Rushby [7], could provide integrity for the security enforcing domains and control all inter-domain communication.[8]

4. SUMMARY

This paper has reviewed the concepts introduced in the Anderson Report, and the evolution and interpretation of those concepts in the TCSEC, ITSEC, and ISO Access Control Framework. The evolution in the understanding of these concepts is useful. However, the fundamental concept of separation — a necessary feature of trust technology — was overlooked in the evolutionary process. Moreover, as the trust IT arena pushes forward, the reality of multiple policies being enforced by the same system, while recognized as a necessity by more and more developers, has not been addressed during this evolution.

In this paper, we have suggested the use of a domain separation mechanism as a fundamental mechanism of trust technology. We suggest that such a mechanism could be employed to isolate security-enforcing functions as sets of domains to address multiple policies. Such domain separation may also yield positive results if applied for policy-enforcing applications. Further research is required to determine the complete value of using a domain separation mechanism as the foundation for building trusted IT systems.

As previously noted, none of the existing documentation addresses the possibility of multiple policies. This is a difficult issue. There are no tools currently available that support the determination of the impact of multiple policies on a system. Lacking such tools, there is no way to evaluate the interaction of constituent access control policies and assess the points of intersection and/or conflict. One dependency for the development of such tools is the identification of common terminology for expressing policies. While formal languages are available, few policies or models are actually formally specified. The tools that use the formal languages support only single policies and only provide verification support at the current time. Thus, there is no ready solution to addressing multiple policies within a single TCB, though we believe this to be an area in need of significant research.

Both the use of separation and support for multiple policies are issues that need to be incorporated into the evolutionary cycle as more mature interpretations of the Anderson Report are formulated. Future standards and criteria cannot afford to ignore the user's view that multiple, tailored policies are desirable, nor the attendent support for multiple policies offered by domain separation.

LIST OF REFERENCES

[1] European Communities — Commission (EC), 1991, *ITSEC: Information Technology Security Evaluation Criteria*, Office of Official Publications of the European Communities, Luxembourg.

[2] International Organisation for Standardisation, International Electrotechnical Committee, Joint Technical Committee 1, Subcommittee 21, 1991, "Working Draft on Access Control Framework," document number 6188, Draft.

[3] Anderson, J. P., October 1972, *Computer Security Technology Planning Study*, ESD-TR-73-51, Vol. I, AD-758 206, ESD/AFSC, Hanscom AFB, Bedford, MA.

8 It appears that this use of the Domain Separation Mechanism implements a noninterference policy. Further investigation is required.

[4] S. Walker, Personal Communication.

[5] Department of Defense (DoD), 1985, *Department of Defense Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, original version published in 1983.

[6] Branstad, M., C. Pfleeger, D. Brewer, C. Jahl, and H. Kurth, October 1991, "Apparent Differences Between the U.S. TCSEC and the European ITSEC," *Proceedings of the 14th National Computer Security Conference*, Baltimore, MD.

[7] Rushby, J., September 1984, "A Trusted Computing Base for Embedded Systems," *7th DOD/NBS Computer Security Conference*, Gaithersburg, MD.

[8] Rushby, J.M., and B. Randell, April 1983, "A Distributed Secure System" (extended abstract), *Proceedings of the 1983 Symposium on Security and Privacy*, Oakland, CA.

[9] Rushby, J., October 1986, "Kernels for Safety?", *Proceedings of the Safety and Security Symposium*, Centre for Software Reliability, Glasgow, Scotland.

[10] Biba, K. J., April 1977, *Integrity Considerations for Secure Computer Systems,* ESD-TR-76-372, MTR-3153, The MITRE Corporation, Bedford, MA.

[11] Clark, David D., and D. R. Wilson, April 1987, "A Comparison of Commercial and Military Computer Security Policies," *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, Oakland, CA.

[12] Graubart, T. D., October 1989, "On the Need for a Third Form of Access Control," *Proceedings 12th National Computer Security Conference,* Baltimore, MD.

[13] La Padula, L. J., 12 June 1990, "Formal Modeling in a Generalized Framework for Access Control," *Proceedings of the Computer Security Foundation Workshop III*, Franconia, NH.

[14] La Padula, L. J., August 1991, *A Rule-Base Approach to Formal Modeling of a Trusted Computer System*, M91-021, The MITRE Corporation, Bedford, MA.

[15] McCollum, C. J., J. R. Messing, and L. Notargiacomo, May 1990, "Beyond the Pale of MAC and DAC — Defining New Forms of Access Control," *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, Oakland, CA.

[16] Abrams, M. D., J. Heaney, O. King, L. J. La Padula, and I. M. Olson, October 1991, "Generalized Framework for Access Control: Towards Prototyping the ORGCON Policy," *Proceedings 14th National Computer Security Conference*, Baltimore, MD.

[17] Abrams, M. D., K. W. Eggers, L. J. La Padula, and I. M. Olson, October 1990, "Generalized Framework for Access Control: An Informal Description," *Proceedings 13th National Computer Security Conference*, Baltimore, MD.

[18] Olson, I. M., and M. D. Abrams, December 1990, "Computer Access Control Policy Choices," *Privacy & Security*, Elsevier.

[19] National Computer Security Center (NCSC), 31 July 1987, *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria,* NCSC-TG-005, Version-1, NCSC.

[20] Adkins, M., G. Dolsen, J. Heaney, and J. Page, October 1989, "The Argus Security Model," *Proceedings 12th National Computer Security Conference*, Baltimore, MD.

[21] Hosmer, H., June 1991, "The Multipolicy Machine, A New Paradigm for Multilevel Secure Systems," *Standard Security Label for GOSIP, An Invitational Workshop*, NISTIR 4614, National Institute for Standards and Technology.

[22] Abrams, M. D., and M. V. Joyce, 1992, *On Multiple Access Control Policies In A Trusted Computing Base,* unpublished, The MITRE Corporation, McLean, VA.

[23] International Organisation for Standardisation, 1988, *Information Processing Systems – Open Systems Interconnection – Security Architecture*, International Standard 7498-2.

METAPOLICIES II

Hilary H. Hosmer
Data Security Inc.
58 Wilson Road
Bedford, MA 01730

ABSTRACT

Metapolicies, or "policies about policies", may become a powerful concept for developing the large, complex, and interrelated trusted systems that military, commercial and non-profit organizations need today. Metapolicies provide a framework for clarifying policies, for organizing security properties, and for successfully coordinating security policies and subpolicies. In a TCSEC unified-policy environment, metapolicies may be implicit, embedded, and fixed. In a multipolicy system of multiple, perhaps contradictory policies, metapolicies must become explicit and support policy flexibility. This paper explores implicit metapolicies, metapolicies for policy conflict resolution, and other characteristics and functions of metapolicies.

INTRODUCTION

RELATED WORK

This paper consolidates and expands the metapolicy concept we introduced in "Integrating Security Policies"[1], "The Multipolicy Machine: A New Paradigm For Multilevel Secure Systems"[2],, and "Metapolicies I".[3] It also builds on the security framework papers of John Dobson[4], John McDermid and Ernest Hocking of York, England[5]. on the work of the Policy Workbench team at George Mason University[6], on Holden's management policy work[7], on Moffet and Sloman's research into policies[8], and on the Generalized Framework for Access Control (GFAC), a rule-based approach started by Planning Research Corporation (PRC)[9] and expanded by MITRE[10].

PRELIMINARY DEFINITIONS

Policy-making is a human enterprise, integrating many complementary, contradictory, fuzzy, and changing human values. A **policy** is a set of constraints established by an accepted authority to facilitate group activity. A good policy provides guidelines for application scope, standard practice, exceptions, and change over time. An organization normally has many policies, which sometimes come into conflict. **Subpolicies** are policies which contribute to a broader policy. **Security policies** are the plans of an organization to meet its security goals, often generalized as confidentiality, integrity, and availability. Those portions of the organization security policies which are implemented on the computer are called **automated security policies**[11]. Automated security policies traditionally comprise identification and authentication (I&A) policies, access control policies, audit policies, and backup and recovery policies, among others. Automated policies may be administratively-imposed[12] (legally-mandated, organizationally-required, derived from standards, or driven by evolving *ad hoc* computer norms) or user-controlled (discretionary access control).

METAPOLICIES DEFINED

Metapolicies are policies about policies. They make the rules and assumptions about policies explicit rather than implicit and coordinate the interaction of multiple policies.

A Metapolicy may be either:

1) a set of rules about a single policy, specifying what kind of policy it is, what elements make up the policy, the universe or domain to which the policy applies, who has the authority to change the policy, the procedure for changing policies, and the relationships to subpolicies;

or

2) a set of rules for coordinating the enforcement of multiple policies, specifying, for example, the order in which multiple policies are enforced, and which results have precedence if a conflict in policies occurs.

This paper will show that metapolicies can:

Describe policy structure and interrelationships;
Control policy additions or modifications;
Coordinate policies and subpolicies.

Metapolicies provide several benefits. They clarify security policies, including underlying assumptions, interactions and integration. They increase policy flexibility, allow multiple policies in a system, create a framework for complex security policies, permit diverse and rich security policies, and permit tailored policy systems to match the legal and organizational policies of diverse clients. They also have drawbacks. They are an unproven concept, add complexity to already complex systems, and may make trusted systems take even more time and money to design, develop and implement.

EXPLORING METAPOLICIES

MAKING IMPLICIT METAPOLICIES EXPLICIT

Explicit metapolicies are not a new concept. For example, social clubs and other organizations often have a set of rules for the club and a separate set of by-laws which describe how the club rules are established and changed. The club rules are the club's policy, and the by-laws are the club's explicit metapolicy. However, security metapolicies are usually implicit and built into both hardware and software.

For example, in SCTC's LOCK system, a user who wants to get access to an object must meet the combined access control requirements of three separate policies: a standard MAC policy, a type enforcement policy, and an integrity policy.[13] A Boolean AND operation built into the hardware combines the results of the user's request to access the object under each of the three policies. This built-in metapolicy can't be changed to some other combination policy, such as OR or XOR which might be desirable when inherent conflicts like those identified by Burns.[14] are encountered between security policies. Immutability provides assurance, but the user loses flexibility. This is unfortunate because the application owner is often the only appropriate person to choose which policy should have precedence when policies conflict[13].

Most security metapolicies today are as invisible and immutable as in the LOCK example. To illustrate we uncover nine metapolicies[15] implicit in a liberal adaptation (see Figure 1) of one of the best known security policies, Bell and LaPadula's (BLP) Simple Security Property. To illustrate what kinds of metapolicies are needed, we name the policy elements and add real-life concerns and constraints.

```
POLICY NAME:                    No Read Up
POLICY TYPE:                    Access Control Subpolicy
AUTHORITY:                      Secretary of Defense
CHANGE PROCESS:                 Consultation with Armed Services

APPLICATION DOMAIN
    This policy applies at all times to all computer systems containing
    USA military classified data.

DOMAIN INTERFACES
    This policy may relate to NATO and SEATO in accordance with Reg. X.

INFORMAL STATEMENT OF POLICY
    No user or process representing a user may read data at a higher
    classification level than the user's clearance level.
        EXCEPTIONS
            Users or processes with downgrade privilege are excepted.
        RELATED AUDIT POLICIES
            Security-relevant events must be auditable.
            Attempted violations must be auditable.
            Any violation must be audited and alarmed.
            Every use of downgrade privilege must be audited.
        OTHER RELATED POLICIES
            Users must identify themselves and be authenticated at login.
        PRECEDENCE RULES
            This policy has priority over any other access control policy.

FORMAL STATEMENT OF POLICY
        S  Subject: User, process, active entity
        O  Object: File, passive entity
        CR  CleaRance
        CL  CLassification

    May_Read (S, O)
        Begin
          If CR(S) >= CL(O)     \check simple security\
              then May_Read = YES
          Else
          If Downgrade(S)= YES    \downgrade privilege?\
              then May_Read = YES.
          If Audit (May_Read) = YES
              then write audit record.
        End

OTHER
```

Figure 1. Adaptation of the Simple Security Property To Show Metapolicies

The No-Read-Up access control policy illustrates several points. To be implemented on an AIS a security policy needs explicit: a) scope, b) description, c) structure, d) interrelationships, e) control, and f) formal and informal renderings for clarity for both machines and people. Metapolicies, or policies about policies, provide a framework for these usually implicit elements. The metapolicy components implicit in Figure 1 are:

1. A Policy Description Metapolicy The names of the elements, the structure of the presentation, and the conventions of the policy description (such as both informal and formal policy statements) constitute a framework that gives meaning to the elements of the policy much the way that data description metadata gives meaning to raw data elements in the database world. The GMU Policy Workbench team calls this descriptive framework a "policy schema".

Policy Description	Data Type	Length	Criticality	Req. Signer	Modifier
Policy Name	Alphanumeric	20	30	Scty DOD	SSO
Policy Type	Alphanumeric	5	30	None	SSO
Authority	Alphanumeric	30	50	Scty DOD	SSO
Start Date	Date	6	20	President	SSO
Expiration Date	Date	6	25	President	SSO
Informal Model	Alphanumeric	900	20	None	SSO
Formal Model	ZED	1500	40	Sys Manager	SSO
etc.					

Figure 2A. Policy Description Metapolicy

The simplified metapolicy example in Figure 2A names the elements of the policy, provides data type and length information (important for automated policies), and provides control information for each element. Signers and modifiers indicate who may approve changes to the policy element and who may actually modify/add/delete the element in an automated security system. The criticality code indicates how much impact a change in the policy element will have on the rest of the policy and/or the security of the system. Many other policy data items could be included.

2. A Policy Relationship Metapolicy A relationship between policies is described by a metapolicy which specifies the policies involved, whether the relationship is hierarchical or collegial, how important the relationship is to the security of the system, which policy is executed first, whether they are always executed together, which has precedence in case of conflict, and who created and who can change the relationship. Figure 2B illustrates.

Policy Relationship Metapolicy	Policy 1	Policy 2
Policy Names	MAC	DAC
Relationship (Parent/Child/Colleague)	Collg	Collg
Execute (With/Before/After/Not)	Before	After
Precedence Level in this relationship	100	50
Criticality of relationship	80	
Creator of relationship	X. Jones	
Authorized Modifiers of relationship	Sec. DOD & SSO	
etc.		

Figure 2B. Policy Relationship Description Metapolicy

A generalized standard format for policy descriptions and policy relationships, designed to be flexible like the proposed GOSIP standard label, would facilitate the integration of multiple policies within and across systems.

3. A Policy Constraint Metapolicy This metapolicy specifies the constraints put on the policy. These could include restrictions on the application domain, environmental constraints like whether or not we are at war, time limitations on the policy due to expiration date, phased processing, or different day and nighttime policies[16]. Other constraints might exempt certain users or roles from the policy, or require the policy to be executed in combination with another policy.

4. A Subpolicy Interaction Metapolicy. The policy in Figure 1 explicitly operates in concert with many subpolicies, such as login, audit, downgrade, label interpretation, and application-specific access control policies. To emphasize these often overlooked subpolicy relationships, such as those shown in Figure 3 below, their many possible interactions are defined in the Subpolicy Interaction Metapolicy.

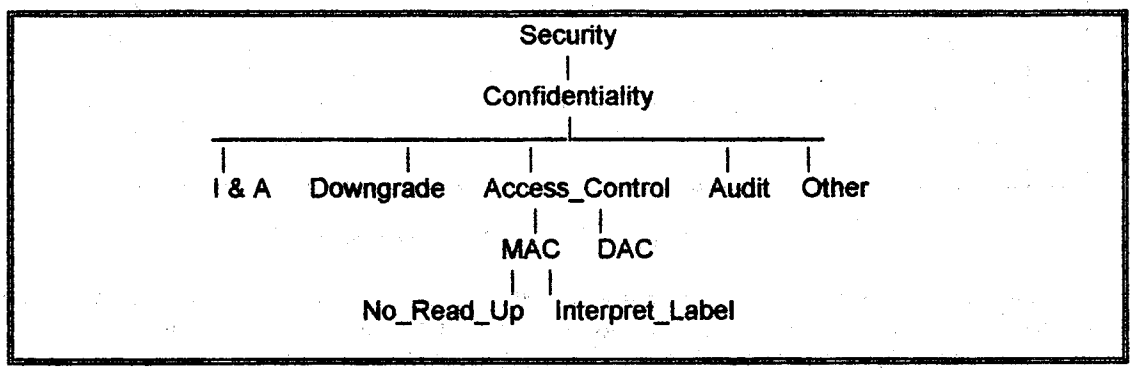


Figure 3. Hierarchical Subpolicy Structure

Subpolicy interactions could also be a described with the Policy Relationship Metapolicy, since each subpolicy is a policy in its own right.

5. The Organization Control Metapolicy This metapolicy describes who owns the policy, who created the policy and when, the policy expiration date, whether the policy can be renewed or modified, and what the processes are for distribution, renewal and modification. It may also include policy assurance status, legal status, the source of the policy (eg. Executive Order 12356) and in what documents the policy appears. The organization control metapolicy is critical for policy flexibility and for policy conflict resolution. The importance of this control function is underscored by McDermid and Hocking[17] who identify three pages of control objectives for security policies, and by Moffet and Sloman who see explicit control as essential in the commercial sector where dominant authority is not as clear as in the military.[18] An example organization control policy is included in Figure 4.

6. Automated Information System (AIS) Metapolicy Formal and informal models tend to be abstract and omit implementation details. This metapolicy is needed to absorb all the additional detail needed to describe and control the implementation of the policy in an automated information system. This detail might include constraints on implementation mechanisms, requirements for configuration management and audit, and other computer-oriented information.

7. Site-Specific Metapolicy The Multipolicy Paradigm gives the local SSO the ability to exert much more control over system policy than is now possible. The site-specific metapolicy lets the SSO describe and control administrative or domain-wide policies entered at the user site.

8. Multipolicy Coordination Metapolicy In a multipolicy machine, a security policy must interact with one or more security policies which may all claim precedence. This metapolicy coordinates multiple security policies in accordance with the user's priorities and tradeoffs. This may be a complex metapolicy, with many levels, domains, and implementation forms. An example of policy coordination by metapolicies appears in Figure 5.

POLICY NAME: Organizational Control
POLICY TYPE: Metapolicy
AUTHORITY: AIS Policy Center
CHANGE PROCESS: Two SSOs with written approval from AIS Policy Center

UNDERLYING POLICY
- Policy Name: No Read Up
- Source of policy: Executive Order 123456
- Legal Status of policy: Mandated by federal law

POLICY PEDIGREE
- Owners: DoD
- Creator: Defense AIS Security Policy Center
- Date Created: 1962
- Expiration Date: 1992
- Authors: John Smith, Sarah Jones
- Reviewers: MITRE, Aerospace

ASSURANCE
- Policy Criticality: High
- Assurance Level of Policy: B3
- Policy Evaluator: Commercial Evaluation Center #3

APPROVAL PROCESS
- Final Authority: President of the USA
- Approving Organizations: US Department of Defense Army, Navy, Marines, Air Force, Coast Guard, Defense Intelligence Agency, DARPA, Joint Chiefs of Staff
- Approval Sequence: Approving organizations give their approval in parallel, then it goes to the President

POLICY IMPLEMENTATION
- Effective Date: 1963
- Application Scope: Applies to all USA classified data
- Oversight Committee: Joint Chiefs of Staff Committee 234

RENEWAL
- Renewal authorization: President of USA
- Renewal terms: 3 to 10 years
- Renewal Process: Service and JCS approval

MODIFICATION
- Authorization for modification: President of USA
- Policy modifier: US AIS Security Policy Center
- Process for modification: Review by all services
- Last Date Modified: 1985

DISTRIBUTION: (Unlimited/Limited/Controlled)
Unlimited

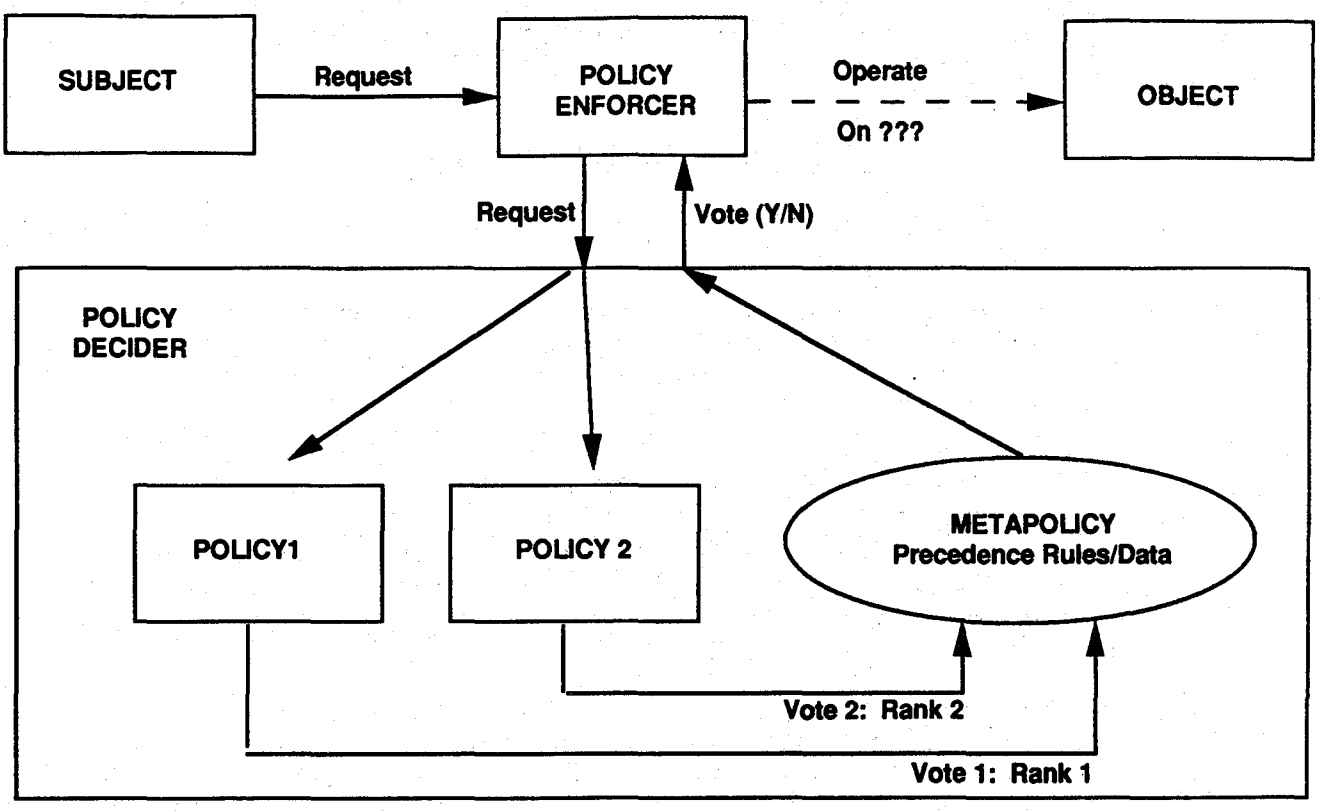
PUBLICATION DATA:
- Publisher: DoD Publications Center
- Document: Military AIS Security Policy

POLICY USED IN:
- Government Documents: DOD 654321, AF 802-456
- Commercial Hardware: all MLS products
- Commercial Software: all MLS products

Figure 4. Organizational Control Metapolicy

Figure 5. Metapolicy for Multipolicy Coordination[19]

(Figure taken from "The Multipolicy Paradigm", also in these proceedings).



1) When a 'Subject' wishes to operate on the 'Object', the request must be mediated by the 'Policy Enforcer'.

3) The Enforcer routs the request to the Policy Decider which, based upon the data's policy domain codes, distributes portions of it to various Policy Decision-Makers (labeled Policy 1 and Policy 2).

4) Using rules and decision data to evaluate the request, each Policy Decision-Maker sends a 'Yes', 'No', 'Don't Care', 'Undecided' or a number on a continuum (fuzzy logic) vote to the Metapolicy. A rank indicating the importance of the policy goes along.

5) The votes of all the individual policies (Vote 1 and Vote 2 in this example) are combined by the Metapolicy according to its rules and data and the ranks of the policies.

6) The resulting 'Yes' or No' vote is sent back to the Policy Enforcer which then permits or denies the requested operation.

9. Domain Interface Metapolicy Metapolicies are most critical at security policy domain interfaces. For example, if data labelled for one policy domain must be transferred to another policy domain, there will be a policy about policies, or metapolicy, describing the rules for any automated transfer.

In summary, the Simple Security Property has several implicit metapolicies:

- Policy Description Metapolicy
- Policy Constraint Metapolicy
- Organization Control Metapolicy
- Site-Specific Metapolicy
- Domain Interface Metapolicy
- Policy Relationship Metapolicy
- Subpolicy Interaction Metapolicy
- Automated Information System Metapolicy
- Multipolicy Coordination Metapolicy.

CHARACTERISTICS OF METAPOLICIES

The primary objective of metapolicies seems to be to provide control for the organization, for the AIS, and for the security subsystem.

Every security policy appears to have multiple metapolicies. Metapolicies may coordinate many policies. Implicit metapolicies aren't obvious, and there seems to be an art to making the implicit explicit. There are general rules which hold for all situations, and there are sets of rules which apply to certain situations, but not others. There are bilateral agreements which apply only to two parties. Metapolicies must be able to handle all these possibilities: general rules, group or subset rules, and individual rules.

Metapolicies differ vastly in scope and significance and must be well-structured so that it is easy to see what prevails over what. There will be metapolicies about metapolicies as well as policies. Their implementation may vary in complexity from a single value to elaborate modular and layered data structures. Layers may correspond to the layers of the organization, the layers of the computer system or the layers of security policies.

Like any security policy, all metapolicies must be protected from tampering or interference. Changes or additions must be audited. If stored on hardware or firmware, validation of the correct operation of the hardware or firmware must be provided. In short, all the requirements that apply to any Trusted Computing Base (TCB), apply to metapolicies, since those portions which are implemented in a computer system become a component of the TCB.
Security policies evolve over time. Metapolicies, by providing control data like that in Figure 4, provide support for conscious and careful evolution of a variety of security policy forms.

Changes to metapolicy are security-relevant events, just like changes to policy. They should be implemented only by the system security officer or a representative. Major changes might require the two-man rule. All changes to metapolicies should be audited. In systems that share duplicate policies, there should be periodic (but surprise) configuration audits to verify that there have been no unauthorized changes to the shared policy.

METAPOLICY FUNCTIONS

From the discussion above, it is clear that metapolicies can play several key functions in trusted systems. They describe policies, support control of policies, provide policy flexibility, coordinate policies, enforce tradeoffs between competing policies, and aid the interfacing of policy domains. They also can help standardize the policy formats and improve the interchangeability of policies.

METAPOLICY DEVELOPMENT

Knowledge engineering strategies from expert systems will be useful in making the implicit explicit and in defining policy and metapolicy rules. Formal modeling will provide rigorous analysis of policies, metapolicies, and their interactions. Graphic techniques, such as those used by database designers, can help visualize the metapolicy relationships, groupings, and interactions.

METAPOLICY IMPLEMENTATION

Metapolicies can be implemented in many ways. The Policy Workbench group recommends an active data dictionary for representing policies [and metapolicies] and their relationships. Moffet and Sloman recommend an object-oriented database model for policies [and metapolicies] because of the hierarchical structure, inheritance and ability to define interactions between policy objects. The GFAC group and their PRC predecessors recommend an expert-system style rule-base for the policy [and metapolicy rules]. Eric Leighninger of DRC recommends the Backus Naur Form (BNF) for policy representation. Our earlier work[1] sketched several possible architectures for multipolicy networks and distributed systems. Regardless of the implementation method, it is clear that security policies require complex data structures, rather than simple embedded rules.

CONCLUSION

Security policies are much more sophisticated than originally thought when the DOD security policy model dominated the field. Many researchers have looked for a framework which would help develop and manage these sophisticated policies. Metapolices, or policies about polices, are an intuitive approach which builds on what is already implicit in any security policy implementation. The Multipolicy Paradigm with metapolicies promises a conceptually elegant framework for managing multiple and sophisticated security policies. However, there is much research and development to be done before the promise becomes a reality.

This paper surveyed the Metapolicy concept. It illustrated that metapolicies can clarify underlying policy assumptions and relationships and facilitate expression of the variety, richness, and multiplicity of security policies. It illustrated how metapolicies permit the controlled interaction of policies and subpolicies, making complex policy systems possible.

Metapolicies, or 'policies about policies', may become a powerful concept for coordinating the multiple, complex, and interrelated security policies that military, commercial, and non-profit organizations need today.

ACKNOWLEDGEMENTS

The Air Force's Electronic Systems Division sponsored this research with a Small Business Innovative Research Phase I grant under contract number F19628-91-C-0157. Grace Hammonds of AGCS, Rae Burns of MITRE and three anonymous reviewers made many constructive suggestions. J. Bret Michael of IDA and GMU critiqued the paper and contributed recent technical work. Marshall Abrams of MITRE provided unpublished work in the multipolicy area. Eric Leighninger of DRC supported our work in multipolicies and outlined research directions. Victoria Ashby of MITRE and Rowena Chester of Martin Marietta created the opportunity to present the multipolicy concept in two ACM SIGSAC workshops at two major conferences. Thank you all.

REFERENCES

[1] Hosmer, Hilary H. , "Integrating Security Policies", *Proceedings of the Third RADC Database Security Workshop, June 5-7, 1990, Castile, N.Y.*, MITRE MTP 385, May 1991.

[2] Hosmer, Hilary H., "The Multipolicy Machine: A New Paradigm for Multilevel Secure Systems", Standard Security Label for GOSIP: An Invitational Workshop, Gaithersburg, MD, NISTIR 4614, June 1991.

[3] Hosmer, Hilary H., "Metapolicies I", ACM SIGSAC Data Management Workshop, San Antonio, TX, December 1991.

[4] Dobson, John, "Specifying Access to Information: Who, Why, and What", University of Newcastle upon Tyme, July 1991.

[5] Dobson, J.E. and J.A. McDermid, "Security Models and Enterprise Models", *Database Security, II Status and Prospect,* North Holland, 1989.

Dobson, John, and McDermid, John, "A Framework for Expressing Models of Security Policy", *IEEE Computer*, July 1989.

[6] Sibley, Edgar, James B. Michael, and Richard Wexelblat, "An Approach to Formalizing Policy Management", *CECOIA2-Proceedings of the 2nd International Conference on Economics and Artificial Intelligence*, Pergamon Press, Oxford, England, 1991.

Sibley, Edgar H., James Bret Michael, and Richard Wexelblat, "Use of an Experimental Policy Workbench: Description and Preliminary Results", *Proceedings of the IFIP TC11.3 5th Working Conference on Database Security,* North-Holland, Amsterdam, 1991.

[7] Holden, D.B. "An Exploration of the Nature of Management Policy", ESPRIT/5165/harw/ T2.1/1_0, AEA Industrial Technology, Harwell Laboratory, Oxfordshire, UK 5 February 1991.

[8] Moffett, Jonathan D. and Morris S. Sloman, "The Source of Authority For Commercial Access Control", *IEEE Computer,* February 1988.

Moffett, Jonathan D. and Morris S. Sloman, "The Representation of Policies as System Objects", *Proceedings of the Conference on Organizational Computer Systems (COCS'91)* Atlanta, Georgia 5-8 November, 1991.

[9] Page, John, Jody Heaney, Marc Adkins, Gary Dolsen, "Evaluation of Security Model Rule Bases", *Proceedings of the 12th National Computer Security Conference,* Baltimore, Maryland, 1989.

[10] Abrams, M.D., K.W. Eggers, L.J. La Padula, and I.M. Olson, "A Generalized Framework for Access Control: An Informal Description," *Proceedings 13th National Computer Security Conference*, Washington, D.C. October 1990.

[11] Sterne, Daniel, "On the Buzzword Security Policy", *Proceedings of the 1991 IEEE Computer Security Symposium on Research in Security and Privacy*, Oakland, CA, May 20-22, 1991.

[12] Abrams, Marshall, Nuance Forum, Dockmaster, 1991.

[13] Haigh, T. ACM SIGSAC presentation, NCSC Conference, Washington, D.C. October 3, 1991.

[14] Burns, R. K., "Referential Secrecy", Proceedings of the IEEE Computer Security Symposium, Oakland, CA, 1990

[15] Six were originally included in "Metapolicies I"[3].

[16] The policy constraint metapolicy is similar to but broader than the GFAC model's "Access Control Context data".

[17] McDermid, John, Ernest Hocking, "Security Policies for Integrated Project Support Environments", *Database Security III: Status and Prospects*, North Holland, 1989.

[18] Moffet, J. and M. Sloman, "The Source of Authority for Commercial Access Control", IEEE Computer, February 1988.

[19] The diagram and description combine our diagram of metapolicies resolving policy conflicts from [1] with Dr. Marshall Abrams' unpublished policy conflict resolution process for access control policies which uses a voting approach from LaPadula, Leonard J. "A Rule-Base Approach to Formal Modeling of a Trusted Computer System", M91-021, August 1991.

A Model for the Measurement of Computer Security Posture

Lee Sutterfield
Todd Schell
Gregory White
Kent Doster
Don Cuiskelly

Abstract

The Air Force is incorporating the use of Statistical Process Control (SPC) methods within the Air Force Command, Control, Communications and Computer Systems Security Program (C4 Systems Security Program). The Air Force C4 Systems Security Program is based on the premise that significant improvement in security posture is unattainable using a traditional authoritarian approach to policy making, procedures development, education and awareness training, etc. Instead of driving such actions only through regulation, they must be based on a clear understanding of the quality of security posture in the field. The key to the measurement of the quality of security posture is the use of Statistical Process Control. SPC theory states that any activity can be clearly defined as a process and the effectiveness of that process can be most effectively improved through the careful measurement and incremental modification of parts of the process. The measurement of security posture has never been considered an easy task. Computer security, in particular, involves so many variables it's hard to know where to begin in the definition of security posture, much less the measurement of it. This paper provides the beginning of an on-going effort to define computer security posture in meaningful, measurable terms. The taxonomy presented has twelve levels of computer security concerns. Though we use the term "computer security" throughout the paper it is important to understand that the taxonomy is intended to account for all security considerations involved in the three main security disciplines most directly affecting the security posture of computer systems--Communications Security (COMSEC), Computer Security (COMPUSEC), and Emanations Security (TEMPEST). The model provides a framework in which the total security environment for a system may be examined in an organized way.

BACKGROUND

The use of Statistical Process Control (SPC) to improve the quality of products and services is well documented. SPC has been successfully used in the manufacturing and service industries in Japan and increasingly within the United States and Europe. A full discussion of SPC is beyond the scope of this paper. However, some perspective on the use of SPC in C4 Systems Security within the Air Force is needed to understand the need for the taxonomy presented in this paper.

SPC has traditionally been used to improve processes within well defined boundaries such as a particular manufacturing process or a well defined administrative process within an organization. Candidate processes for the application SPC usually have a few common characteristics. First, the process can be clearly diagramed in a flow chart showing all actions and actors. Second, a "process owner" is easily identifiable. The process owner is the highest authority within the organization that controls key components of the process. The process owner takes on the ultimate responsibility for the improvement of the process. Third, any organizational boundaries that must be crossed in improving the process must be under the ultimate control or at least strong influence of the process owner. These three characteristics of a candidate process for the application of SPC become increasingly difficult to handle as the complexity of the process and the size of the organization grows.

It is this complexity factor that makes the application of SPC to Air Force C4 Systems Security difficult. C4 Systems Security is complex. The overall security posture of a computer system will involve issues and resources ranging from security policy and procedures; organizational structure; education and awareness; physical and environmental security; connectivity issues; access controls; operating system trust; hardware and media control issues; personnel security and others. Which areas are key to good security posture? Are they quantifiable? In addition, the size of the Air Force as an organization makes it difficult to establish a common process for computer security for all Air Force organizations.

Another issue of concern with the use of SPC is that it has traditionally been most effectively used when the management culture of the organization is built on what is referred to as Total Quality Control or Total Quality Management philosophy. SPC by nature requires a long-term approach to the constant improvement of products and services. Without the foundation of Total Quality Management (TQM) principles throughout the organization it is difficult to effect any long-term, continuous improvement in quality. The Air Force, along with many other government organizations and agencies, has embarked on the TQM path. The adoption of TQM philosophy

throughout the Air Force will make the success of our use of SPC methods more likely. However, the use of SPC methods without the benefit of the wide practice of other TQM ideas can still provide a more organized and thorough approach to improving security posture.

The incorporation of Statistical Process Control (SPC) methods within the Air Force C4 Systems Security Program has generated a number of initiatives. The heart of the program is the C4 Systems Security Vulnerability Reporting Program (CVRP). Under this program we are more clearly defining a number of processes and developing special tools, expertise and procedures to improve those processes. As examples, we have developed the Automated Risk Evaluation System (ARES) and the C4 System Security Management System (CMS) to standardize and improve the the risk analysis and accreditation process, the Electronic Security Engineering Teams to improve the system certification process, and the Air Force Computer Emergency Response Team (AFCERT) to control the vulnerability and incident handling processes. We have also built Electronic Security Survey Teams (ESSTs) to act as the primary resource to objectively measure organizational security posture in the field. The security posture model presented in this paper is the basis for the measurement strategy and techniques used by the ESSTs to quantify and measure C4 Systems Security Posture.

All of the initiatives and implications of the use of SPC in the Air Force C4 Systems Security Program are beyond the scope of this paper. We have addressed these issues in other writings. This paper focuses on the definition of the "Air Force C4 Systems Security Posture Model."

SECURITY POSTURE DEFINITION

C4 Systems Security Posture is currently defined as the instantaneous sum of all security policy, procedures, guidance, technical and administrative resources, operational activities and general system use practices that provide for the confidentiality, integrity, and uninterrupted service of C4 systems and the information processed and controlled by those systems. In this definition is implied all the security issues as defined by the traditional communications security (COMSEC), computer security (COMPUSEC), and emanations security (TEMPEST) disciplines. It is further implied that the one constant of security posture is that it will change. Because of the obvious role of people in the security process and the ever changing technologies involved, it is clear that security posture is continuously variable. The good security posture of today will be the poor security posture of tomorrow.

During the last two years we have experimented with various methods and resources to measure security posture in the field. We formed adhoc teams to measure the C4 systems security posture of organizations. These teams found numerous security problems in every organization visited. Despite a fairly large and active C4 Systems Security Program within the Air Force, the C4 systems security posture of AF organizations is not what it should be. Considerable effort has been expended to build and implement strong education and awareness efforts, organized R&D efforts, vulnerability and incident handling capabilities, and strong policy, procedures and guidance. We have many people at many levels doing their best to build good security posture and yet their efforts don't seem to be effective.

During these initial efforts to measure security posture we learned important lessons that are changing the way the C4 Systems Security Program is conducted in the Air Force. Though a discussion of all of those lessons are beyond the scope of this paper, one lesson which stands out among the rest is the need to accurately measure security posture in the field. The C4 Systems Security Posture Model presented here forms the foundation of the effort to systematically measure security posture throughout the Air Force C4 community.

Air Force C4 Systems Security Posture Model

The model attempts to identify the major factors that provide layers of security for any C4 system. It is important to understand that this model is not an attempt to define generic security features and requirements for computer operating systems or related security technologies. It is an attempt to provide a standard framework in which the total security environment for a system may be examined in an organized way.

The hierarchical layers start at the organizational level with ***Level 1*** and end within the system at ***Level 12***. It is important to note that the model is not based on current regulatory or procedural requirements within the Air Force. The model is based on the major factors that comprise consistent, high quality security posture. The most important of the levels are of course at the top. Without the top levels it will be impossible to build and maintain consistent security over the long-term. However, the practical implementation of good security practice occurs at the lower levels.

A cursory examination of the model may lead some to suggest that several levels in the model should be combined to simplify the model,

AIR FORCE C4[1] SYSTEMS[2] SECURITY POSTURE MODEL

Level 1: System[3] Mission. A statement describing the purpose and function of the system, the sensitivity and criticality of its data and processes, and the security requirements and environment of the system.

Level 2: Security Policy. A written interpretation of all existing laws, policies, regulations, and guidance as they apply to the security of the system, its processes, data, and products.

Level 3: Security Organizational Structure. A formalized hierarchy of specialized security management positions, each having detailed responsibilities, clearly defined authorities, and appropriate span of control.

Level 4: Security Implementation Procedures. A compilation of local regulations, OIs, operational plans and procedures, which, if properly implemented will ensure compliance with the stated security policy.

Level 5: Security Education, Training & Awareness. A formal program that ensures all security management personnel are trained in their respective disciplines and all system users are aware of their security responsibilities.

Level 6: Physical and Environmental System Protection. The facility characteristics and operational procedures used to control physical access to the system, its processes, data and products, and to protect the system from environment hazards.

Level 7: System Connectivity Controls. The communications architecture and topology designed to control electronic linkage to the system.

Level 8 : System Access Controls. All identification and authentication control mechanisms used to control logical access to the system, its processes, data, and products.

Level 9: System Administration Controls. All actions taken to ensure optimal use and integrity of system security features and security hardware/software.

Level 10: Storage Media Controls. All actions taken and resources available to control the access to, protect the integrity of, ensure the availability of, and the proper disposal of storage media associated with the operation of the system.

Level 11: Accountability Controls. All activities and resources that consistently collect, record, trace, and resolve all actions that have security implications.

Level 12: Assurance. The sum of all actions and resources that provide a degree of trustworthiness and credibility to all aspect of system operations.

1 **C4 Systems Security: The protection afforded command, control, communication, and computer systems in order to preserve the availability, integrity, and confidentiality of the systems and information contained within the systems. Such protection is the integrated application of the three C4 security component programs: COMSEC, COMPUSEC, and TEMPEST executed in liaison with OPSEC, INFOSEC, personnel security (PERSEC), physical security (PHYSEC), and other security disciplines as necessary.**

2 **Systems Security Posture: The sum of security measures, processes, and procedures applied to a system (see note 3) to ensure availability protection from compromise, and integrity for the system, its processes, data, and products.**

3 **System: As used in this definition, refers to a single accreditable entity; i.e., a stand-alone PC, a LAN, a WAN, a Mini or Mainframe computer and its connected terminals.**

especially when viewed strictly from a computer security point of view. However, the goal of the model is to provide a framework around which we can build a mechanism to measure security posture in relation to existing COMSEC, COMPUSEC and TEMPEST requirements. Rationale and examples are provided for each level.

Level 1: System Mission. *A statement describing the purpose and function of the system, the sensitivity and criticality of its data and processes, and the security requirements and environment of the system.*

Security must begin with a thorough understanding of the purpose of the system. All operational requirements must be thoroughly understood before security requirements can be identified. The nature of the system mission will drive the fundamental security policy for the system. This is not to be confused with an organizational mission. The system mission should be the specific role that the system plays in the overall organizational mission.

The operational requirements should be clearly articulated in written form. Though this may not be a regulatory requirement it is consistent with good security practice. We have separated system mission from security policy in recognition that the primary purpose of C4 systems is to provide operational capabilities to organizations. Security is a necessary ingredient to ensure the quality of that capability, but it is not an end product of the system itself.

Level 2: Security Policy. *A written interpretation of all existing laws, policies, regulations, and guidance as they apply to the security of the system, its processes, data, and products.*

This level will establish all security requirements for the system. It is usually accomplished as an integral part of a formal risk analysis which, ideally, should take place early in the acquisition cycle of the system.

Level 3: Security Organizational Structure. *A formalized hierarchy of specialized security management positions, each having detailed responsibilities, clearly defined authorities, and appropriate span of control.*

Considering the complexity of today's communications-computing environment it is vital that any large organization have a security administrative structure in place. In the Air Force that structure is comprised of positions such as the Major Command Computer Security Manager, Computer System Security Officer, Terminal Area Security Officer, TEMPEST

Control Officer, etc. Maintaining good security posture requires continual dissemination of information such as policy, procedures, special technical information, etc. In addition, persons in these positions should have clearly defined and supported authority to make security related decisions. Without a formal security organizational structure there is little hope of improving security posture throughout the organization.

Level 4: Security Implementation Procedures. *A compilation of local regulations, operating instructions, operational plans and procedures, which, if properly implemented will ensure compliance with the stated security policy.*

Implementation procedures for security for a given system will be based on general guidance provided by higher organizational levels special local requirements. Security implementation procedures are the backbone of continued quality security posture. These procedures are often complex and seldom stated in a single security document.

Level 5: Security Education, Training and Awareness. *A formal program that ensures all security management personnel are trained in their respective disciplines and all system users are aware of their security responsibilities.*

Much effort has gone into education and awareness in the past and more effort will be needed in the future. In the past this training focused on good general security practices and regulatory requirements. In the future, the nature of the training must become more specific about the quality of security posture. For example, we constantly tell computer users and administrators not to connect computers to networks without implementing certain security features and yet we continue to find unauthorized connectivity throughout the Air Force. We have **told** users not to connect the systems without precautions but we haven't provided hard data that clearly shows **why** they shouldn't. In keeping with good TQM practices, future education, training and awareness efforts in the Air Force will be based on data, not opinion or regulations. Our experience has shown that when users are given data to support policy and procedures they usually modify their practices. This model will provide the framework in which such security posture data will be collected. The objective measurement of the quality of training itself is difficult to perform. The effectiveness of the 5th Level will probably be best judged by the measurement of Levels 6-11.

Level 6: Physical and Environmental System Protection. *The facility characteristics and operational procedures used to control physical access to*

the system, its processes, data and products, and to protect the system from environmental hazards.

This level has traditionally been the easiest to examine. Standards for the physical protection of systems are well known and easily measured. It is at this level that the nature of points of measurement may begin to differ for the three disciplines (COMSEC, COMPUSEC, TEMPEST). For example, the issue of control space for TEMPEST may be different than the issue of controlled entry spaces for COMPUSEC.

Level 7: System Connectivity Controls. *The communications architecture and topology designed to control electronic linkage to the system.*

We have distinguished between System Connectivity Controls and the System Access Controls of Level 8. One might be inclined to combine the two but there is advantage in keeping them separate. For example, classified systems within the Air Force have been afforded an extra level of security protection through the issuance of strict policy regarding the connectivity of such systems to public communications systems. The computer system itself will still need considerable system access controls to maintain good security posture within its environment. However, a large part of the security risk to these systems has been mitigated by a conscious decision not to connect them to unclassified communications systems. The measurement of connectivity controls should be relatively simple technically. However, fully describing the connectivity of systems within operational environments is becoming increasingly difficult for system owners. This level will require considerable energy to completely characterize.

Level 8: System Access Controls. *All identification and authentication control mechanisms used to control logical access to the system, its processes, data, and products.*

The variety of System Access Controls have increased dramatically in the last few years. Measuring the effectiveness of these mechanisms, at least at a basic level, will be an area of emphasis for the ESSTs. Though access controls don't answer all security concerns, they provide the backbone of good computer security in relation to today's environment of extensive connectivity.

Level 9: System Administration Controls. *All actions taken to ensure optimal use and integrity of system security features and security hardware/software.*

The role of the system administrator and computer system security officer is crucial to the security of multi-user systems. System administration controls encompass all actions by the system administrative and security personnel and users to ensure that the security posture of the system is optimized at all times. This includes such actions as the control of maintenance accounts, user accounts, passwords, privileges, security review procedures for product outputs, account application procedures, etc. The status of various security features and the security discipline maintained on the system is probably the most measurable of the 12 layers. We have made considerable progress in the development of automated tools to measure Levels 8 & 9.

Level 10: Storage Media Controls. *All actions taken and resources available to control the access to, protect the integrity of, ensure the availability of, and the proper disposal of storage media associated with the operation of the system.*

Again, this is an area of concern that is fairly easy to quantify, although it may take considerable resource time to do so. It includes such issues as the practice of labeling magnetic media, transfer of data via the "sneakernet", and the regular practice of media backup and storage, etc. The combination of Levels 7, 8, 9 & 10 can form the core of the most practical measures of daily security posture in the field.

Level 11: Accountability Controls. *All activities and resources that consistently collect, record, trace, and resolve all actions that have security implications for the system.*

Accountability controls go beyond the traditional audit trails within an operating system to include administrative and special actions capabilities to resolve suspected incidents. For example, if a security incident occurs, i.e. a system "cracker" or "malicious logic" incident, are all of the system resources such as audit trails and other activity indicators usable to help identify and remedy the situation. In addition, are the procedures in place to handle such an incident, to include vulnerability and incident reporting procedures and isolation and containment procedures and tools, etc. The key is to measure the ability to resolve anomalies that have security implications. These factors may also include basic education, awareness and training issues. Levels 1 thru 10 form the basis for pro-active security. If those levels are accomplished effectively the need for Level 11 capabilities would be minimal. Level 11 is really focused on the reactive aspect of security.

Level 12: Assurance. *The sum of all actions and resources that provide a degree of trustworthiness and credibility to all aspects of system operations.*

Assurance lies at the heart of the "trusted system" concept. This level of course includes the trusted system issues, but it is intended to describe all actions that can be taken to provide a "measured" level of assurance as opposed to an "evaluated" level of assurance in the trusted system sense. Level 12 includes such activities as security test and evaluation, use of statistical process control techniques at the local level to monitor security on a specific system, as well as the use of trusted system software.

CONCLUSION

The use of the above security posture model has already proven beneficial in the Air Force's attempt to quantify security posture in the field. The limited data collected and lessons learned so far have already influenced changes in long-standing procedures and guidance that used to be accepted as effective. As the status of our security posture becomes clearer, we expect to identify unexpected areas for improvement that can be affected by changes in policy, procedures or guidance. As time progresses, we expect, with the aide of tools such as the C4 Systems Security Posture Model to bring security posture in the field under tight control and thereby enhance all aspects of C4 systems support to the operations community.

Deming, W. Edwards, *Out of the Crisis,* MIT_CAES, 1986.

Ishikawa, Kaoru, *Guide to Quality Control,* Asian Productivity Organization, Tokyo, Japan, 1982.

Walton, Mary, *Deming Management at Work,* Putnam Publishing Group, NY, 1991.

A MODEL OF RISK MANAGEMENT IN THE DEVELOPMENT LIFE CYCLE

Captain Charles R. Pierce

Air Force Cryptologic Support Center (AFCSC/SRP)
San Antonio Texas 78243-5000

ABSTRACT

Computer security risk management has traditionally been viewed as a process for determining what protection measures are required for computer resources. Its primary protection target has been computer facilities and mainframe computers housed within them. This is a result of the origin of the risk management mandate, Office of Management and Budget (OMB) Circular A-71 [1], subsequently reiterated in Circular A-130 [2]. The primary targets of these directives were facilities and large systems. Many government agencies and vendors have since developed risk analysis models and products to meet the OMB defined goals. Many of these were designed to meet Federal Information Publication (FIPS) Pub 65 [3] direction, whose goal was to implement the OMB direction. But the computer world has changed to where major computer systems can now consist of mostly microcomputers and local area networks. There are no "computer facilities" for these systems since they reside in normal office environments. Some of their resources, such as telephone lines, are not proprietary to the system's owning organization. In the meantime computer security threats have expanded from the original traditional resource protection view (power, fire, etc.) to more active and hostile environments (hackers, viruses, industrial espionage). In addition to these traditional targets, newer applications, such as computers embedded within other systems (processes controllers, communications switches) have become more vulnerable. The conventional risk management model centered on resource protection no longer meets the protection needs for system developers or users. To broaden protection, models developed by the Department of Defense have extended risk management to address information sensitivity. DoD's interest is acute when classified information is the property at risk. However, DoD models have yet to integrate risk management fully into the system development life cycle, only making suggestions as to when to perform a certain function, such as security test and evaluation (ST&E). This paper proposes a life cycle risk management model for managing risks throughout a system's life cycle. It focuses on the life cycle phases when the system is built because that is when to best counter risks by building in protection measures. However, since it is a life cycle process it must include risk abatement activities that address continuous change and the system's eventual disposal. Because systems currently exist at various stages of development or operation, the model must provide entry points throughout for those who did not begin risk management at the beginning.

RISK MANAGEMENT TODAY

As exercised today in most locations, risk management consists of three phases; some form of risk analysis, a technical certification that the system's implemented security features meet stated requirements, and an accreditation to operate the system in a particular environment. Various risk analysis methodologies exist, both quantitative or qualitative or combinations of both techniques. Quantitative methodologies usually base loss expectations on some form of monetary values such as annualized loss expectancy [3]. Qualitative methods, used where exact costs are less easily determined, apply such values as high, medium, or low. Often a mix of the two methods is used. Monetary values are used for hard resources, e.g., equipment and buildings, and qualitative values are used for softer assets, such as information sensitivity and personnel experience. The depth of detail in a risk analysis can vary based on many factors ranging from the system's size to the levels of information's sensitivity. Certification is nominally based on risk analysis results for the hardware, software, and facilities. The certification's value can vary widely. To certify that all security requirements are met does not ensure that all requirements were properly stated or that a system is indeed resistant to penetration or failure. If secure communications were not initially required, the certification will not address them and the risk to hacker penetration will also not be addressed. Accreditation is the manager's decision to operate the system in its proposed environment based on the certifications. Unfortunately, in many cases the accrediting authority has not been involved in security features development until the accrediting decision is required.

One of the greatest problem with risk analysis is determining the level of effort to pursue in its performance. Few methods or tools easily adapt to all sizes and types of systems. A powerful tool that would model a large multilevel network of diversely configured nodes could be overkill for a suite of unconnected office support microcomputers. Likewise a facility asset protection tool would not address the fuzziness of multiple information sensitivity levels. These and other factors complicate the decision on the level of security features to require, such as the level of Trusted Computing Base (TCB) [4]. DoD Directive 5200.28 [5] provides a matrix for determining a required TCB level based on information sensitivity and user clearance levels. However, the matrix does not adequately consider environmental factors nor is it easily applicable where there is no classified information or outside the DoD environment. Nor does it address such issues as data integrity or service assurance at a level comparable to sensitivity. There is recent work in process on developing certification standards that addresses these issues [6][7]. All these ambiguities in risk analysis and resulting certifications propagate into accreditation, forcing decisions based possibly on inaccurate information or invalid assumptions.

One of the major difficulties with current risk analysis technologies are that they typically take a "slice-in-time" view of a systems. The result is a snapshot of the analyzed system. Analyzed security requirements are viewed as non-flexible fait accompli, not as results of the risk analysis process itself. Updating and maintaining risk analysis results usually means performing the entire analysis again. This is because neither the models or tools allows for sensitivity analysis (from the accountant's point of view) of computations or easy reentry, since the tools must be used serially. The proposed model addresses these and other issues.

THE DEVELOPMENT LIFE CYCLE

As defined by the revised DoD Instruction 5000.2 [8], the acquisition life cycle consists of five phases; Concept Exploration and Definition, Demonstration and Validation, Engineering Manufacturing and Development, Production and Deployment, and Operations and Support. Each phase is preceded by a milestone that reviews and completes the preceding phase and begins next phase. Each milestone requires the system's program manager to preform certain actions and produce defined documents, such as the Test and Evaluation Master Plan (TEMP), for milestone decision authority review and approval. A program does not advance until all actions required for the previous phase have been completed or adequately addressed. The development life cycle is preceded by requirements development and their documentation in a Mission Need Statement (MNS) that defines a mission deficiency and an Operational Requirements Document (ORD) defining broad performance requirements for the proposed system.

The DoDI 5000.2 addresses risk management from a programmatic point of view, i.e., cost, schedule, and technology, but not from a threat to information resources direction. Information resource threats are to be considered in the MNS prior to Milestone 0, Concept Studies Approval (before beginning Concept Exploration and Development), and must be specifically assessed before Milestone I, Concept Demonstration Approval (before Demonstration and Validation). The depth of effort a program office must exert to protect the system must be "consistent with mission requirements and cost-effectiveness." What this depth actually is not further defined, but using common sense is implied. The program's protection countermeasures are to be addressed by a multidisciplinary approach that addresses risks, environments, and the developmental technology used during the life cycle. The protection should include a time phased plan to transition the security concept and countermeasures as the life cycle progresses. MIL-STD-1785 [9] is to be used for establishing a system security program prior to Milestone II, Development Approval (before Engineering Manufacturing and Development).

Thus, it seems that DoDI 5000.2 addresses those activities, e.g., threat assessments, countermeasures development, cost-benefit analyses, etc., that are usually considered a part of risk analysis and management. Detailed study however reveals that the implied point of view is not that of a comprehensive program aimed at reducing computer security risks. The described security program leans more toward a typical "security police" type of effort that aims to protect the development program itself, not to provide protection mechanisms internal to the developing system. The program manager is left to his own devices to integrate DoDI 5000.2's various security pronouncements into an organized, concise statement of requirements and to implement an effective program to meet them.

The risk management model's goal is to provide a methodology to meet the program manager's need to comprehensively address systems risks throughout the life cycle, from requirements definition through system operation to replacement or disposal. Because the model addresses requirements generation, its life cycle begins before the system development life cycle as defined in the DoD publications.

THE MODEL

The model begins by expanding typical information sensitivity and system criticality assessments to increase the number of security factors to consider. Sensitivity involves issues in information and personnel security and, to some extent, physical security. Criticality (defined here as service assurance and data or system integrity) is increasingly being addressed by computer security practitioners, but is most often rolled into such traditional system issues as reliability and maintainability. It is a primary issue for systems such as those requiring nuclear surety and medically related life support surety. The model expands the factors to consider for risk analysis to include the full environment of security requirements. This includes communications security (COMSEC), emanations security (TEMPEST), physical security, personnel security, administrative security, procedural security, etc. In other words, all the securities included in the "security chain." This integrated requirements analysis provides the input for developing a system level security policy. An initial, "quick look," risk analysis and macro level security policy is encouraged, but not required, when developing the system's need requirements, i.e., the MNS. This "quick look" should be used to determine the feasibility of the proposed security features. The first full blown iteration of an integrated risk analysis will be performed after system development is approved, early in concept exploration and definition. This analysis should be performed by the system's Program Management Office (PMO) together with those who initially require the system and its eventual users. The goal is to determine the earliest possible allocation of security requirements to security disciplines or features. This will result in proposed TCB levels, clearance levels, facility

security features, etc., that can then be allocated to requirements and specifications.

Guidance currently exists for many of the subanalyses required to implement the fully integrated security requirements analysis. Both Air Force [10] and DoD [11] guidelines provide for initial TCB level determinations. As discussed before these levels need to be "massaged" for other environmental considerations. Once sensitivity levels are determined, DoD policy is specific on what clearance levels are needed and the related basic physical security requirements. TEMPEST and COMSEC measures are more easily determined due to fairly recent specific guidance development. [12] Criticality requirements are not as easily translated into trust or security "levels," but there are efforts in this area at the Air Force [13] and the National Institute of Standards and Technology (NIST) [14]. This portion of the model could be greatly enhanced by automated tools to implement requirements analysis.

The model requires the PMO or procuring agent to establish a Security Working Group (SWG) if the system's size or complexity warrants it. (It's imperative to mention at this time that the model provides the flexibility to adapt to system size, complexity, or intended application. It's equally adaptable for large scale automated information systems or embedded applications.) The SWG is the PMO's main focal point for all security relevant activities.

The primary deliverable products in concept development consists of various plans (e.g., certification, accreditation, test, maintenance, logistics support, etc.), proposed concepts of operation, and an initial risk analysis. The model requires early delivery of the plans. They, along with the risk analysis, must be reviewed and updated as necessary at each program milestone. The idea is to build currency and security into the system's risk management as the system develops, just as you would do for any technology advancements that the development could absorb.

As system's development progresses from concept definition into demonstration and validation or prototype development, the system's requirements allocation to security features becomes more concrete. TCB levels firm up, and TEMPEST and COMSEC features begin to be engineered into the design. Now is the time to complete detailed, formal risk analyses and to deliver their results. The model requires particular items to be developed and delivered during the life cycle phases for PMO evaluation (Figure 1). Risk analysis results become part of the program manager's required presentation to the milestone decision authority before the development can proceed to the next life cycle phase. In essence, the milestone decision authority is exercising Designated Approving Authority (DAA) authority.

PHASE	ACTION	PRODUCT
Milestone 0 - Concept Studies Approval		
Concept Exploration and Definition (CE&D) (Phase 0)	Initiate Risk Management Identify Security Requirements Identify Trusted Computing Base (TCB) Develop Security Test Plans Develop System Security Plan Develop System Security Policy Identify Security Focal Points	User Validated Security Requirements Draft Specifications Include Initial Risk in Security Plan Source Selection Criteria Security Policy, to Include CONOPS
Milestone I - Concept Demonstration Approval		
.	.	.

Figure 1. Sample Development Actions and Products

At this time documentation supporting risk management can easily be contracted to the system developer, if contracting is the acquisition strategy. The Air Force [15][16] and National Computer Security Center (NCSC) [17] provide guidance and specification language for risk management support. The model provides data requirements and timing requirements for those deliverables. Once the PMO produces final system requirements and a system security policy, risk management documentation can become a set of contract deliverables. If the system is sufficiently small, or exists of primarily off the shelf components, it's possible for the PMO or SWG to perform all required risk management activities and develop all required documents. The size of the products and effort expended on them is still a difficult determination, but AFSSMs 5024 and 5028 provide guidance on what you can expect for TCB products.

The relative effort expended on plans, assessments, etc., diminishes as the development progresses into engineering manufacturing and development. The risk management focus now centers on implementing and testing security measures, evaluating their effectiveness, and defining residual risks. Properly planning and coordinating security features delivery with system development is essential if they are to be tested concurrently with other system features, a necessity if these are truly integrated, "system" tests. The model requires security testing to not stand out as a separate function. Since the model generated security requirements early in the life cycle, just as other operational requirements were stated in the ORD, these

requirements should be tested the same as other features are tested for functionality. The only difference is that "security" is their operational mission. Concurrent testing also provides a better capability to assess security measures impact on other system functions.

As the program development moves to production versions, the model emphasizes completing remaining risk analysis functions (i.e., testing) and progressing to system certification and operational accreditation. The program manager certifies the system, including all security relevant features under his control, as having met security requirements, with or without residual risks. These residual risks could have been revealed as the system developed, will be discovered as not having been addressed in early analyses, or will simply be risks the security measures cannot fully counteract. In any case the DAA for the system's user will either accredit the system in its final configuration or grant short term interim accreditation while residual risks are addressed. At accreditation the system becomes fully integrated with its operating environment. The DAA must not only consider the program manager's certification but also that for the operating facility, if one exists. Remember, back in requirements analysis, certain security requirements will have been allocated to countermeasures other than those within the computer system itself. This can include the bulk of environmental security features, such as resources protection, contingency plans, and off-site backup and storage. The model also considers these facility security features. The model also provides for maintaining risk management itself, since risk management is a life cycle process mated to the system development process.

MODEL FLEXIBILITY

A major feature of the model is its adaptability to various incarnations of computer systems. The model cares not if the system is a standalone mainframe, a widespread network of multiple nodes, a local area network of off-the-shelf microcomputers, or a network of embedded computer resources within a tactical weapon system. The model is keyed to security and the life cycle, not the system specifics. It is a requirements translation model not based on specific architectures. Therefore, if all of its steps are not required for the development or a particular architecture, they may be omitted (with DAA approval). If a fully developed TCB is not required (such as for embedded systems), the model can be fulfilled without the TCB evaluation, and certification still achieved. There are minimum documentation requirements, because they are risk management requirements, e.g., plans, threat definitions, test results, etc. In fact, taking the model's basic boilerplate, tailoring it to the particular system's development requirements, and adding roles and responsibilities for performing its activities will produce a simplified security or certification/accreditation plan. The resulting plan is

probably incomplete, and requires adding such features as the time phasing for activities, since these depend on the program's phasing. Flexibility is not without potential pitfalls however. Too many adjustments and a failure to fully appreciate their effects can lead to an incomplete and insufficient effort. As with today's current planning efforts, it behooves the PMO to have an outside review of the proposed plan. The model provides for an independent verification and validation activity for reviewing activities.

MODEL ADAPTABILITY

Much as the model is flexible about the level of effort to spend on risk management, it is also adaptable as to how risk management fits in the development life cycle. It defines sufficient reentry points where you could begin to apply risk management in a development. Ideally you should begin with requirements analysis, but if you find you're lagging behind, there are loops in the model to get risk management to a reasonable point. In fact, the last activity indicated by the model is to recycle back into itself appropriately. Since recertifications and reaccreditations are required at various times or upon significant events, its imperative that the model provide for reentry and reperformance. Therefore, the model never closes until system disposal. This allows it to be applied to those systems that have completed development and are actually in operation.

MODEL DEVELOPMENT

Major portions of the model have been already been developed. The integration of computer security into the development life cycle is depicted in AFSSM 5010 [18], produced by the Air Force Cryptologic Support Center. This particular document does not focus specifically on risk management, but instead takes a rather high level view of integrating risk analysis and various security features into the development life cycle. One of its primary attributes is a depiction of key DAA interface points in the life cycle.

Further work in the model and a DoD certification standard is being done by a working group under the auspices of the Joint Logistics Commanders' Information Systems Security (INFOSEC) Management Panel (IMP) that includes the integration of risk management into the life cycle. The risk management model is being proposed as subset of the more encompassing standard for certification. The progress to date on the certification standard and its subelements is:

1. Identify Key Concepts and Objectives (Completed January 1992)

2. Provide Standard Definitions (Completed January 1992)

3. Outline Certification-Related Processes

   Risk Management (Completed September 1992)

   Evaluation (Completed April 1992)

   Accreditation (Completed September 1992)

4. Recommended Standards and Policy (First Draft September 1992)

5. Identify Needed Implementation Resources (Completed September 1992)

6. Identify Training Requirements (Completed September 1992)

It is intended that the risk management model remain an informal model when completed. A strictly formal model would indicate a desire for strict adherence to it in an implementation. This would lead to attempts to formally apply the model during a program development, thereby negating the model's intended adaptability and flexibility.

REFERENCES

1. Office of Management and Budget, Circular A-71, Transmittal Memorandum No. 1, Security of Federal Automated Information Systems, 27 July 1978.

2. Office of Management and Budget, Circular A-130, Management of Federal Information resources, 12 December 1985.

3. Guideline for Automatic Data Processing Risk Analysis, FIPS Pub 65, National Institute of Standards and Technology, August 1979.

4. Department of Defense Trusted Computer System Evaluation Criteria, 26 December 1985.

5. Department of Defense Directive 5200.28, Security Requirements for Automated Information Systems (AIS), 21 March 1988.

6. Pierce, C.R., Standardized Certification, Proceedings of the 14th National Computer Security Conference, 1991.

7. Pierce, C.R., Standardized Certification - Progression, Proceedings of the 15th National Computer Security Conference, 1992.

8. Department of Defense Directive 5000.2, Defense Acquisition Management Policy and Procedures, 23 February 1991.

9. MIL-STD-1785, System Security Program Management Requirements.

10. AFSSI 5100, The Air Force Computer Security (COMPUSEC) Program, 2 June 1992.

11. CSC-STD-003-55, Computer Security Requirements, 25 June 85.

12. NTISSC 7000, (S)TEMPEST Countermeasure of Facilities(U), 17 October 1988.

13. AFSSM 5029, Trusted Critical Computer System Certification Criteria (Draft), 1 April 1992.

14. Wallace, Dolores R., D. Richard Kuhn, and John C. Cherniavsky, Proceedings of the Workshop on High Integrity Software; Gaithersburg, MD; Jan 22-22, 1991, NIST SP 500-190, National Institute of Standards and Technology, 1991.

15. AFSSM 5024, Security in Acquisitions (Draft), 1 April 1992.

16. AFSSM 5028, Acquisition Language for Critical Systems (Draft), 10 June 1992.

17. NCSC-TG-24, An Introduction to Procurement Initiators on DoD Computer Security Requirements (Draft), 25 October 1991.

18. AFSSM 5010, Computer Security in the Acquisition Life Cycle (Draft), 1 April 1992.

*U.S. G.P.O.:1992-625-512:60546