

*NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY/
NATIONAL COMPUTER SECURITY CENTER*

17th NATIONAL COMPUTER SECURITY CONFERENCE

**October 11-14, 1994
Baltimore Convention Center
Baltimore, Maryland**



PROCEEDINGS VOLUME 1

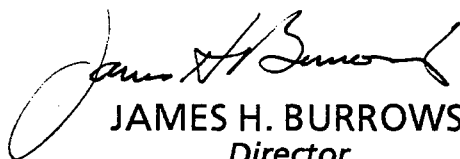
**Communicating our Discipline
Strategies for the Emerging Information Infrastructures**

Welcome!

The National Computer Security Center (NCSC) and the Computer Systems Laboratory (CSL) are pleased to welcome you to the Seventeenth Annual National Computer Security Conference. There is a new sense of urgency in the U.S. and abroad to achieve protection for the rapidly evolving information infrastructures. This year's program is designed to provide you information on the exciting new opportunities and the latest security technology. We believe the conference will stimulate a copious exchange of information and promote a solid understanding of today's information security issues and solutions.

The program tracks have been established to serve a wide range of interests from highly technical R&D projects to user oriented management and administration topics. Clearly, network security is a high priority topic. The opening and closing plenary sessions will highlight various dimensions of the security challenges in emerging information infrastructures. Papers and panel sessions will address a broad spectrum of network security subjects including: security architecture, internet security, firewalls, multilevel security (MLS) products, MLS system certification and accreditation, and security management. There will be a report on the progress and status of the Common Criteria and efforts for international harmonization. Risk management is a topic of increasing interest in today's difficult economic environment. As in the past, a number of tutorials will be given to introduce attendees to various information security topics and product areas.

We hope the networking conducted at the conference, the presentations and these proceedings will provide you with insights and ideas you can apply to your own information security endeavors. We encourage you to share the ideas and information acquired this week with your peers, your management, and your customers. Through this process we will enhance the security of our information systems and networks and build a strong foundation to meet tomorrow's challenges.


JAMES H. BURROWS
Director
Computer Systems Laboratory


PATRICK R. GALLAGHER, JR.
Director
National Computer Security Center

Referees

Professor Sushil Jajodia

Dr. Steven Kent

Leslee LaFountain

Steven LaFountain

Paul A. Lambert

Dr. Carl Landwehr

Dr. Theodore M.P. Lee

Steven B. Lipner

Teresa Lunt

Frank Mayer

Dr. Catherine Meadows

William H. Murray

Dr. Peter Neumann

Steven Padilla

Marybeth Panock

Nick Pantiuk

Donn Parker

Dr. Gopal Ramanathan

Philip M. Roney

Dr. Ravi Sandhu

Marvin Schaefer

Daniel Schnackenberg

Steven Skolochenko

Bill Smith, CISSP

Dr. Stuart G. Stubblebine

Patricia Toth

Captain Charles Tracey, USAF

Dr. Chii-Ren Tsai

Kenneth R. VanWyk

John Wack

Grant Wagner

Major Glenn Watt, USAF

Howard Weiss

Roy Wood

Thomas E. Zmudzinski

George Mason University

Bolt, Barenek & Newmann

National Security Agency

National Security Agency

Motorola Incorporated

Naval Research Laboratory

Independent Consultant

Trusted Information Systems

SRI International

The AEROSPACE Corporation

Naval Research Laboratory

Deloitte & Touche

SRI International

SPARTA, Inc.

The MITRE Corporation

Grumman Data Systems

SRI International

The MITRE Corporation

Computer Sciences Corporation

George Mason University

ARCA Systems

Boeing Defense and Space Group

U.S. Department of Justice

Defense Information Systems Agency

USC Information Sciences Institute

National Institute of Standards and Technology

Joint Staff, Pentagon

Citicorp Global Information

Defense Information Systems Agency

National Institute of Standards and Technology

National Security Agency

U.S. STRATCOM

SPARTA, Inc.

National Security Agency

Defense Information Systems Agency

Awards Ceremony

6:00 p.m. Thursday, October 13
Convention Center, Room 317

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 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.

The Computer Security Division at NIST provides validation services to test vendor implementations for conformance to security standards. NIST currently maintains validation services for three Federal Information Processing Standards (FIPS): FIPS 46-2, Data Encryption Standard (DES); FIPS 113, Computer Data Authentication; and FIPS 171, Key Management Using ANSI X9.17. During this award ceremony, NIST presents "Certificate of Appreciation" awards to those vendors who have successfully validated their implementation of these standards.

With the reaffirmation of the Data Encryption Standard as FIPS 46-2 in 1993, DES can now be implemented in software, as well as hardware and firmware. To successfully validate an implementation for conformance to FIPS 46-2, a vendor must run the Monte Carlo test as described in NBS (NIST) Special Publication 500-20. The Monte Carlo test consists of performing eight million encryptions and four million decryptions, with two encryptions and one decryption making a single test.

Vendors test their implementations for conformance to FIPS 113 and its American National Standards Institute (ANSI) counterpart, ANSI X9.9, Financial Institution Message Authentication (Wholesale). This is done using an electronic bulletin board system. Interactive validation requirements are specified in NBS (NIST) Special Publication 500-156, Message Authentication Code (MAC) Validation System: Requirements and Procedures. The test suite is composed of a series of challenges and responses in which the vendor is requested to either compute or verify a MAC on given data using a specified key which was randomly generated.

Conformance to FIPS 171 is also tested using an interactive electronic bulletin board testing suite. FIPS 171 adopts ANSI X9.17, Financial Institution Key Management (Wholesale). ANSI X9.17 is a key management standard for DES-based applications. The tests are defined in a document entitled NIST Key Management Validation System Point-to-Point (PTP) Requirements. The test suite consists of a sequence of scenarios in which protocol messages are exchanged under specified conditions.

We congratulate all who have earned these awards.

17th National Computer Security Conference

Table of Contents

Refereed Papers

RESEARCH AND DEVELOPMENT, TRACK A

Testing Intrusion Detection Systems: Design Methodologies and Results from an Early Prototype	1
Nicholas Puketza, Biswanath Mukherjee, Ronald A. Olsson, Kui Zhang, University of California, Davis	
A Pattern Matching Model for Misuse Intrusion Detection	11
Sandeep Kumar, Eugene H. Spafford, Purdue University	
Artificial Intelligence and Intrusion Detection: Current and Future Directions	22
Jeremy Frank, University of California, Davis	
A Three Tier Architecture for Role-Based Access Control	34
Ravi S. Sandhu, Hal Feinstein, SETA Corporation	
Using THETA to Implement Access Controls for Separation of Duties	47
Rita Pascale, Joseph R. McEnerney, Odyssey Research Associates	
Implementing Role Based, Clark-Wilson Enforcement Rules in a B1 On-Line Transaction Processing System	56
Barbara Smith-Thomas, AT&T Bell Laboratories; Wang Chao-Yeuh, Wu Yung-Sheng, Institute for Information Industry, Taiwan	
Virtual View Model to Design a Secure Object-Oriented Database	66
N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, K. Yazdanian, ONERA/CERT, France	
Achieving Database Security Through Data Replication: The SINTRA Prototype	77
Myong H. Kang, Judith N. Froscher, John McDermott, Oliver Costich, Rodney Peyton, Naval Research Laboratory	
The Sea View Prototype: Project Summary	88
Teresa F Lunt, Peter K. Boucher, SRI International	
Towards a Formal Verification of a Secure and Distributed System and its Applications	103
Cui Zhang, Rob Shaw, Mark R. Heckman, Gregory D. Benson, Myla Archer, Karl Levitt, Ronald A. Olsson, University of California, Davis	
Making Secure Dependencies over a LAN Architecture for Security Needs	114
Bruno d'Ausbourg, CERT/ONERA, France	

Refereed Papers (Cont'd)

Automatic Generation of High Assurance Security Guard Filters	123
Vipin Swarup, The MITRE Corporation	
Belief in Correctness	132
Marshall D. Abrams, The MITRE Corporation; Marvin V. Zelkowitz, University of Maryland, College Park	
Towards a Privacy-Friendly Design and Use of IT-Security Mechanisms	142
Simone Fischer-Hübner, University of Hamburg	
Using a Semiformal Security Policy Model 2C a C2 Better	153
Marvin Schaefer, ARCA Systems, Inc.;; Gary R. Grossman, Jeremy J. Epstein, Cordant, Inc.	
 ARCHITECTURE AND STANDARDS, TRACK B	
A Taxonomy for Security Standards	165
Wayne A. Jansen, NIST	
The Graphical Display of a Domain Model of Information Systems Security (INFOSEC) Through Semantic Networks: A Description of the INFOSEC Semantic Network for Information Systems Security Engineers	175
Teresa T. Smith, Kathleen V. Dolan, National Security Agency	
A New Attack on Random Pronounceable Password Generators	184
Ravi Ganesan, Chris Davies, Bell Atlantic	
Development History for Procurement Guidance Using the Trusted Computer System Evaluation Criteria	198
Major Melvin L. DeVilbiss, USA, National Security Agency	
Exporting Evaluation: an analysis of US and Canadian criteria for trust	206
Paul A. Olson	
What Color is Your Assurance?	215
David R. Wichers, Joel E. Sachs, Douglas J. Landoll, ARCA Systems, Inc.	
BFE Applicability to LAN Environments	227
Tom Benkart, ACC Network Systems; Dave Bitzer, National Security Agency	
The Architecture of Triad: A Distributed, Real-Time, Trusted System	237
E John Sebes, Nancy Kelem, Terry C. Vickers Benzel, Mary Bernstein, Eve Cohen, Jeff Jones, Jon King, Trusted Information Systems, Inc.; Michael Barnett, David M. Gallon, Roman Zacjew, Locus Computing Corporation	
Constructing a High Assurance Mail Guard	247
Richard E. Smith, Secure Computing Corporation	

Refereed Papers (Cont'd)

APPLICATIONS AND INTEGRATION, TRACK C

Controlled Execution UNIX	254
Lee Badger, Homayoon Tajalli, David Dalva, Daniel Sterne, Trusted Information Systems, Inc.	
Architectures for C2 DOS/Windows-Based Personal Computers, Securing an "Unsecurable" Operating System	264
Jeremy Epstein, Gary Grossman, Frederick Maxwell, Noble Veirs III, Albert Donaldson, Cornelius Haley, Cordant, Inc.	
A Practical Hardware Device for System and Data Integrity as well as Malicious Code Protection	274
T.E. Elliott, Department of National Defence, Canada	
Partitioning the Security Analysis of Complex Systems	283
Howard Holm, National Security Agency	
The Composition Problem: An Analysis	292
Guy King, Computer Sciences Corporation	
Making Do With What You've Got	299
Janis W. Berryman, The Boeing Company; Bruce F. Kennedy, Cubic Applications, Inc.	
Modern Multilevel Security (MLS): Practical Approaches for Integration, Certification, and Accreditation	309
Bill Neugent, Mike Burgoon, Jeanne Firey, Mindy Rudell, The MITRE Corporation	
Applying COMPUSEC to the Battlefield	318
Diane M. Bishop, Stephen R. Arkley, Computer Sciences Corporation	
Security Requirements for Customer Network Management in Telecommunications	327
Vijay Varadharajan, Hewlett-Packard Labs, UK	
Support for Security in Distributed Systems Using MESSIAHS	339
Steve J. Chapin, Kent State University Eugene H. Spafford, Purdue University	
A Technical Approach for Determining the Importance of Information in Computerized Alarm Systems	348
David S. Fortney, Lawrence Livermore National Laboratory, Judy J. Lim, Lim and Orzechowski Associates	

Refereed Papers (Cont'd)

ASAM: A Security Certification and Accreditation Support Tool for DoD Automated Information Systems	358
Loreto Remorca, Jr., William Barr, Secure Solutions, Inc.;	
Robert Zomback, U.S. Army CECOM, Space and Terrestrial Communications Directorate; V. Michael Caputo, MICON Services Company	
A Financial Management Approach for Selecting Optimal, Cost-Effective Safeguards Upgrades for Computer- and Information-Security Risk Management	370
Suzanne T. Smith, Barranca Inc.; Stephen Gale, William J. Malampy, University of Pennsylvania	
 MANAGEMENT AND ADMINISTRATION, TRACK D	
The Electronic Intrusion Threat to National Security & Emergency Preparedness Telecommunications: An Awareness Document	378
Dr. Joseph Frizzel, National Communications System;	
Ted Phillips, Traigh Groover, Booz-Allen & Hamilton, Inc.	
Using Application Profiles to Detect Computer Misuse	400
Nancy L Kelem, Daniel F. Sterne, David I. Dalva, Kenneth M. Walker, Trusted Information Systems, Inc., Debra Anderson, Harold Javitz, Alfonso Valdes, SRI International, Linda L. Lankewicz, Glenn Bell, Spring Hill College	
Can Computer Crime be Deterred?	412
Sanford Sherizen, Ph.D., Data Security Systems, Inc.	
Demonstrating the Elements of Information Security with Threats	421
Donn B. Parker, SRI International	
The Aerospace Risk Evaluation System (ARiES): Implementation of a Quantitative Risk Analysis Methodology for Critical Systems	431
Charles H. Lavine, Anne M. Lindell, Sergio B. Guarro, The Aerospace Corporation	
The Security-Specific Eight Stage Risk Assessment Methodology	441
David L. Drake, Katherine L. Morse, Science Applications International Corporation	
Security Awareness and the Persuasion of Managers	451
Dennis F. Poindexter, Center for Information Systems Security	
The Network Memorandum of Agreement (MOA) Process: Lessons Learned ..	459
William C. Barker, Lisa M. Jaworski, Geroge R. Mundy, Trusted Information Systems, Inc.	
Independent Validation and Verification of Automated Information Systems in the Department of Energy	468
William J. Hunteman, Los Alamos National Laboratory;	
Robert Caldwell, Department of Energy	

Panel Summaries

RESEARCH AND DEVELOPMENT, TRACK A

Fuzzy Security: Formalizing Security as Risk Management	478
Security is Risk Management	480
Ruth Nelson, Chair, Information System Security	
Fuzzy Policies	482
Hilary H. Hosmer, Data Security, Inc.	
Assurance, Risk Assessment, and Fuzzy Logic	483
John McLean, Naval Research Laboratory	
Using Fuzzy Logic in Formal Security Models	486
Sergei Ovchinnikov, San Francisco State University	
Role Based Access Control, Its Structure, Mechanisms and Environment	488
Hal Feinstein, Chair, SETA Corporation	
Role-Based Access Control Position Paper	491
Marshall D. Abrams, The MITRE Corporation	
Role-Based Access Control, A Position Statement	492
Ravi S. Sandhu, George Mason University	
Role-Based Access Control, Position Statement	493
David Ferraiolo, NIST	
Inference Problem in Secure Database Systems	494
Bhavani Thuraisingham, Chair, The MITRE Corporation	
An Inference Paradigm ¹	497
Donald G. Marks, Department of Defense	
The Inference Problem: A Practical Solution	507
Teresa F. Lunt, SRI International	
Security-Oriented Database Inference Detection	510
Thomas H. Hinke, Harry S. Delugach, University of Alabama	
Key Escrowing: Today and Tomorrow	514
Miles E. Smid, Chair, NIST	
The Target System	514
Jan Manning, National Security Agency	
Procedures for Lawful Interception of Telecommunications	514
Mike Glimore, Federal Bureau of Investigation	
Future Considerations for Key Escrowing	514
Dr. Dorothy Denning, Georgetown University	
The Security Association Management Protocol Panel	515
Major Terry Hewitt, USAF, Chair, National Security Agency	
Position Paper	517
James Leppek, Harris Corporation	
Position Paper	518
Dave Wheeler, Motorola	

1. Refereed paper

Panel Summaries

Highlights of the New Security Paradigms '94 Workshop	519
Eric Leighninger, chair	
Formal Semantics of Confidentiality in Multilevel Logic Databases	520
Adrian Spalka, University of Bonn, Germany	
Healthcare Information Architecture: Elements of a New Paradigm	532
Daniel J. Essin, University of Southern California; Thomas L. Lincoln, The RAND Corporation	
Communication, Information Security and Value	554
John Dobson, University of Newcastle, UK	
Fuzzy Patterns in Data--Anomaly Detection	566
T. Y. Lin, San Jose State University	

ARCHITECTURE AND STANDARDS, TRACK B

The Development of Generally Accepted System Security Principles (GSSP), NIST's Approach	581
Marianne Swanson, Chair, NIST	
Viewpoint	581
Will Ozier, ISSA GSSP Committee Chair	
Viewpoint	581
Marianne Swanson, NIST	
Viewpoint	581
Ed Roback, NIST	
Viewpoint	581
Barbara Guttman, NIST	
Product and System Certification in Europe	582
Klaus J. Keus, Chair, BSI, Germany	
Status of European Certification Schemes and Mutual Recognition	582
Angelika C. Jennen, BSI, Germany	
Certification Maintenance under ITSEC	583
Jeremy Wilde, Logica, UK	
Security Evaluations in the Netherlands--An evaluators view on globalization of evaluations	583
Dr. Paul L. Overbeek, TNO Physics and Electronics Laboratory, The Netherlands	
Effectiveness in French Evaluations	583
Laurent Borowski, CR2A, France	
The relation between Correctness and Effectiveness in System Composition	584
Mats Ohlin, Electronic Systems Directorate, Sweden	
Evaluation of Platform Independence	584
Peter Cambell-Burns, Admiral Management Services Limited, UK	
Vendor Assurance vs. 3rd Party Evaluation: A Constructive Approach ..	585
Dr. Heinrich Kersten, BSI, Germany	

Panel Summaries (Cont'd)

New Concepts in Assurance Panel	586
Pat Toth, Chair, NIST	
Viewpoint	586
Lynne Ambuel, National Security Agency	
Viewpoint	586
Deitra Kimpton, CSE, Canada	
Viewpoint	586
Ken Rochon, National Security Agency	
Viewpoint	586
Karen Ferraiolo, ARCA Systems	
New Challenges for C&A: The Price of Interconnectivity and Interoperability	587
Ellen Flahavin, Co-chair, NIST	
Joel Sachs, Co-chair, ARCA	
The Department of Defense Goal Security Architecture (DGSA)	588
W. Timothy Polk, Chair, NIST	
The Department of Defense Goal Security Architecture	588
Richard McAllister, National Security Agency	
The DGSA Overall Transition Strategy	588
Carl Deutsch, National Security Agency	
Security Standards for DGSA-based Architectures	588
Janice Schafer, Defense Information Systems Agency	
DGSA's Applicability to non-DoD Environments	588
Jim Coyle, Booz-Allen & Hamilton	
Multilevel Security--Current Applications and Future Directions	589
Colonel J. Sheldon, USA, Chair, DISA/CISS	
Viewpoint	589
John Wiand, USSOCOM	
Viewpoint	589
Russ Myers, USACOM	
Viewpoint	589
Emily Klutz, USACOM	
Viewpoint	589
Lieutenant Colonel Tom Surface, USPACOM	
Viewpoint	589
Major Kevin Newland, USSPACECOM	
Viewpoint	590
Mr. Paul Woodie, National Security Agency	
Viewpoint	590
Mr. Charles West, DISA	

Panel Summaries (Cont'd)

Prominent Industry-Sponsored Security Architectures Currently Under Development	592
Michael McChesney, Chair, Secure Ware, EGSA	
Viewpoint	596
Roger Schell, Novell	
Viewpoint	598
Bill Dwyer, Hewlett-Packard	
APPLICATIONS AND INTEGRATION, TRACK C	
Can Your Net Work Securely?	599
Peter G. Neumann, Chair, SRI International	
How to Trust a Distributed System	600
B. Clifford Neuman, USC, Information Sciences Institute	
Internet Firewalls	602
John Wack, Chair, NIST	
Proven Detection Tools for Intrusion Prevention	603
Michael Higgins, chair, Defense Information Systems Agency	
MLS System Solutions - A Continuing Debate Among The Critical Players	604
Joel E. Sachs, ARCA Systems, Inc.	
Trusted Systems Interoperability Group	
Stan Wisseman, Chair, ARCA Systems, Inc.	607
Historical Perspective	610
Paul Cummings, Digital Equipment Corporation	
Common Internet Protocol Security Option	613
Ron Sharp, AT&T Bell Laboratories	
Trusted Security Information Exchange for Restricted Environments	614
Charlie Watt, SecureWare	
Trusted Administration Working Group	616
Jeff Edelheit, The MITRE Corporation	
Trusted Applications Working Group	618
Stan Wisseman, ARCA Systems, Inc.	
Government Perspective	619
George Mitchell, NCSC	

Panel Summaries (Cont'd)

NSA Concurrent Systems Security Engineering Support to the MLS TECHNET Program	620
Bradley Hildreth, Chair, National Security Agency	
Viewpoint	624
Mary Mayonado, Eagan, McAllister Associates, Inc	
Viewpoint	626
Teresa Acevedo, Pulse Engineering, Inc.	
Viewpoint	628
Jenny Himes, National Security Agency	
Viewpoint	629
Gregory Wessel, National Security Agency	
Viewpoint	630
Randy Blair, National Security Agency	
Viewpoint	631
Richard White, Air Force Information Warfare Center	
Viewpoint	633
George Hurlburt, Naval Air Warfare Center	
Provisions to Improve Security on the Internet	635
Dr. Harold Highland, Chair, Computers & Security	
Viewpoint	636
Dr. Harold Highland, Computers & Security	
Viewpoint	639
Frederick Avolio, Trusted Information Systems, Inc.	
Viewpoint	641
Dr. Stephen Bellovin, AT&T Bell Laboratories	
Viewpoint	643
Matt Bishop, University of California, Davis	
Viewpoint	645
William R. Cheswick, AT&T Bell Laboratories	
Viewpoint	647
Dr. Jon David, The Fortress	
Viewpoint	650
Colonel Frederick A. Kolbrener, U.S. Army	
Viewpoint	652
A. Padgett Peterson, Martin-Marietta Information Group	

Panel Summaries (Cont'd)

MANAGEMENT AND ADMINISTRATION, TRACK D

Model Information Security Programs	654
Richard W. Owen, Jr., Chair, Office of the Attorney General, Texas	
Viewpoint Academia	654
Stephen J. Green, University of Houston	
Viewpoint Commercial	655
Genevieve M. Burns, Monsanto Company	
Viewpoint Federal	655
Philip L. Sibert, U.S. Department of Energy	
Viewpoint State	656
Jan W. Wright, Information Resources Commission, Florida	
Interdisciplinary Perspectives on InfoSec: Bringing the Humanities into Cyberspace	657
Michel E. Kabay, Chair, National Computer Security Association and JINBU Corporation, Canada	
An Anthropological View: Totem and Taboo in Cyberspace ²	658
Michel E. Kabay, National Computer Security Association and JINBU Corporation, Canada	
Philosophy of Law and InfoSec: Justifying Morality in Cyberspace	669
Virginia Black, Pace University	
Psychology and InfoSec: Improving Compliance with InfoSec Policies ...	675
Percy Black, Pace University	
Military Science and Information Security	681
James P. Craft, Systems Research and Applications Corporation	
Ethical Issues in the National Information Infrastructure	685
Jim Williams, Chair, The MITRE Corporation	
Medical Information Privacy, Current Legislative and Standards Activities ...	688
Marc Schwartz, Support Medical Systems, Inc., Chair	
Viewpoint	690
Robert Gellman, United States House of Representatives	
Viewpoint	692
Molla Donaldson, National Academy of Sciences	
Viewpoint	694
Dale Miller, Irongate, Inc.	
Viewpoint	695
C. Peter Waegemann, Medical Records Institute	
Viewpoint	696
Gerald S. Lang, The Harrison Avenue Corporation	
Privacy and the Handling of Patient Related Information in the Public Swedish Health Care System ³	699
Torleif Olhede, Stockholm University, Sweden	

2. *Refereed paper*

3. *Refereed paper*

Panel Summaries (Cont'd)

Computer Crime on the Internet	713
Christine Axsmith, Esq., Chair, ManTech Strategies Associates	
Viewpoint	714
Donn Parker, SRI International	
Viewpoint	714
Mark Pollitt, Federal Bureau of Investigation	
Viewpoint	714
Ted Chambers, Scientific Computer Support Team	
Viewpoint	715
Barbara Fraser, Carnegie Mellon	
Viewpoint	715
Martin Schoffstall, Performance Systems International	
Viewpoint	716
Mark Fedor, Performance Systems International	
Do You Have the Skills to be a Future INFOSEC Profession	717
Dr. William (Vic) Maconachy, Chair, Center for Information Systems Security	
Computers at Risk Recommendations: Are They Still Valid?	723
Hal Tipton, Chair, HFT Associates	
Viewpoint	724
Will Ozier, Ozier Peterse & Associates	
Viewpoint	724
Earl Boebert, Secure Computing Corporation	
Viewpoint	725
Steve Walker, Trusted Information Systems	
Objectives and Progress of the GSSP Committee	727
TUTORIALS & PRESENTATIONS, TRACK E	
Tutorial Series on Trusted Systems and Operational Security	731
R. Kenneth Bauer, Joel Sachs, Dr. Eugene Schultz, Dr. Gary Smith, Jeff Williams, ARCA Systems, Inc.; Chris Bressinger, DoD Security Institute; Dr. Charles Abzug, LtCdr Alan Liddle, National Defense University	
Security Information for the Asking: The Untapped Information Potential Awaiting the Security Practitioner	733
Viewpoint	733
Kathie Everhart, NIST	
Viewpoint	733
Marianne Swanson, NIST	
Viewpoint	734
Bob Lau, National Security Agency	
Viewpoint	734
Nickilyn Lynch, NIST	

Panel Summaries (Cont'd)

SPECIAL SESSIONS AND DEMONSTRATIONS

International Harmonization: The Common Criteria--Progress and Status	735
Eugene Troy, Chair, NIST	
Security Requirements for Distributed Systems	738
Robert Dobry, Chair, National Security Agency	
The Application of Electronic Groupware Tools to Address IT Security Challenges	739
Dennis Gilbert, Demonstration Coordinator, NIST	
The Learning Track	740
Training Challenges of the 90's	740
Joan Pohly, FISSEA Chair	
Proposed New NIST Training Standards	741
Dorothea deZafra, Public Health Service	
Computer Security Resources that Work	741
Barbara Cuffie, Social Security Administration	
Effective Marketing of the Computer Security Program to Management .	741
Joan Hash, Social Security Administration	
Tools and Methodologies for Delivering Training	741
Janet Jelen, Public Health Service	
Demonstrations on Computer Security Training Tools	741
Anthony Stramella, National Cryptologic School	
Training Events on a Shoestring Budget	741
Sadie Pitcher, Department of Commerce	
Adult Learning and Information Systems Security Training	742
Dr. Eugene V. Martin, Organization and Education Consultant	
Information Systems Professionalism--Professional Development and Certification	742
Richard Koenig, Harold Tipton, International Information Systems Security Certification Consortium	
Computer Ethics for Future Generations	742
Richard Koenig, International Information Systems Security Certification Consortium	

Authors and Panelists Cross Index

Abrams, Marshall D.	132, 491	Coyle, Jim	588
Abzug, Dr. Charles	731	Craft, James P.	681
Acevedo, Teresa	626	Cuffie, Barbara	741
Ambuel, Lynne	586	Cummings, Paul	610
Anderson, Debra	400	Cuppens, F.	66
Archer, Myla	103	Dalva, David I.	254, 400
Arkley, Stephen R.	318	d'Ausbourg, Bruno	114
Axsmith, Christine, Esq.	713	David, Jon	647
Avolio, Frederick	639	Davies, Chris	184
Badger, Lee	254	Delugach, Harry S.	510
Barker, William C.	459	Denning, Dr. Dorothy	514
Barnett, Michael	237	Deutsch, Carl	588
Barr, William	358	DeVilbiss, M. L., Major USA ..	198
Bauer, R. Kenneth	731	deZafra, Dorothea	741
Bell, Glenn	400	Dobry, Robert	738
Bellovin, Stephen	641	Dobson, John	554
Benkart, Tom	227	Dolan, Kathleen V.	175
Benson, Gregory D.	103	Donaldson, Albert	264
Vickers Benzel, Terry C.	237	Donaldson, Molla	692
Bernstein, Mary	237	Drake, David L.	441
Berryman, Janis W.	299	Dwyer, Bill	598
Bishop, Diane M.	318	Edelheit, Jeff	616
Bishop, Matt	643	Elliott, T. E.	274
Bitzer, Dave	227	Epstein, Jeremy	153, 264
Black, Percy	675	Essin, Daniel J.	532
Black, Virginia	669	Everhart, Kathie	733
Blair, Randy	630	Fedor, Mark	716
Boebert, Earl	725	Feinstein, Hal	34, 488
Borowski, Laurent	583	Ferraiolo, David	493
Boucher, Peter K.	88	Ferraiolo, Karen	586
Burns, Genevieve M.	655	Firey, Jeanne	309
Boulahia-Cuppens, N.	66	Fischer-Hübner, Simone	142
Bressinger, Chris	731	Flahavin, Ellen	587
Burgoon, Mike	309	Fortney, David S.	348
Caldwell, Robert	468	Frank, Jeremy	22
Cambell-Burns, Peter	584	Fraser, Barbara	715
Caputo, V. Michael	358	Frizzel, Dr. Joseph	378
Chambers, Ted	715	Froscher, Judith N.	77
Chao-Yeuh, Wang	56	Gabillon, A.	66
Chapin, Steve J.	339	Gale, Stephen	370
Cheswick, William R.	645	Gallon, David M.	237
Cohen, Eve	237	Ganesan, Ravi	184
Costich, Oliver	77	Gellman, Robert	690

Authors and Panelists Cross Index

Gilbert, Dennis	739	Lavine, Charles H.	431
Glimore, Mike	514	Leighninger, Eric	519
Green, Stephen J.	654	Leppek, James	517
Groover, Traigh	378	Levitt, Karl	103
Grossman, Gary	153, 264	Liddle, Alan, LtCdr	731
Guarro, Sergio B.	431	Lim, Judy J.	348
Guttman, Barbara	581	Lin, T. Y.	566
Haley, Cornelius	264	Lincoln, Thomas L.	532
Hash, Joan	741	Lindell, Anne M.	431
Heckman, Mark R.	103	Lunt, Teresa F.	88, 507
Hewitt, Terry, Major USAF	515	Lynch, Nickilyn	734
Higgins, Michael	603	Maconachy, Dr. William (Vic)	717
Highland, Harold, FICS	635, 636	Malampy, William J.	370
Hildreth, Bradley	620	Manning, Jan	514
Himes, Jenny	628	Marks, Donald G.	497
Hinke, Thomas H.	510	Martin, Dr. Eugene V.	742
Holm, Howard	283	Mayonado, Mary	624
Hosmer, Hilary H.	482	Maxwell, Frederick	264
Hunteman, William J.	468	McAllister, Richard	588
Hurlburt, George	633	McChesney, Michael	592
Jansen, Wayne A.	165	McDermott, John	77
Javitz, Harold	400	McEnerney, Joseph R.	47
Jaworski, Lisa M.	459	McLean, John	483
Jelen, Janet	741	Miller, Dale	694
Jennen, Angelika C.	582	Mitchell, George	619
Jones, Jeff	237	Morse, Katherine L.	441
Kabay, Michel E	657, 658	Mukherjee, Biswanath	1
Kang, Myong H.	77	Mundy, Gerorge R.	459
Kelem, Nancy	237, 400	Myers, Russ	589
Kennedy, Bruce F.	299	Nelson, Ruth	478, 480
Kersten, Dr. Heinrich	585	Neugent, Bill	309
Keus, Klaus J.	582	Neuman, B. Clifford	600
Kimpton, Deitra	586	Neumann, Peter G.	599
King, Guy	292	Newland, Major Kevin	589
King, Jon	237	Ohlin, Mats	584
Klutz, Emily	589	Olhede, Torleif	699
Koenig, Richard	742	Olson, Paul A.	206
Kolbrener, F. A., COL USA	650	Olsson, Ronald A.	1, 103
Kumar, Sandeep	11	Ovchinnikov, Sergei	486
Landoll, Douglas J.	215	Overbeek, Dr. Paul L.	583
Lang, Gerald S.	696	Owen, Jr., Richard W.	654
Lankewicz, Linda L.	400	Ozier, Will	581, 724
Lau, Bob	734	Parker, Donn	421, 714

Authors and Panelists Cross Index

Pascale, Rita	47	Thuraisingham, Bhavani	494
Peterson, A. Padgett	652	Toth, Pat	586
Peyton, Rodney	77	Troy, Eugene	735
Phillips, Ted	378	Valdes, Alfonso	400
Pitcher, Sadie	741	Varadharajan, Vijay	327
Pohly, Joan	740	Veirs III, Noble	264
Poindexter, Dennis F.	451	Wack, John	602
Polk, W. Timothy	588	Waegemann, C. Peter	695
Pollitt, Mark	714	Walker, Kenneth M.	400
Puketza, Nicholas	1	Walker, Steve	727
Remorca, Jr., Loreto	358	Watt, Charlie	614
Roback, Ed	581	Wessel, Gregory	629
Rochon, Ken	586	West, Charles	590
Rudell, Mindy	309	Wheeler, Dave	518
Sachs, Joel E. ...	215, 587, 604, 731	White, Richard	631
Sandhu, Ravi S.	34, 492	Wiand, John	589
Schaefer, Marvin	153	Wichers, David R.	215
Schafer, Janice	588	Wilde, Jeremy	583
Schell, Roger	596	Williams, Jeff	731
Schoffstall, Martin	716	Williams, Jim	685
Schultz, Dr. Eugene	731	Wisseman, Stan	607, 618
Schwartz, Marc	688	Woodie, Paul	590
Sebes, E. John	237	Wright, Jan W.	656
Sharp, Ron	613	Yazdanian, K.	66
Shaw, Rob	103	Yung-Sheng, Wu	56
Sheldon, J., COL USA,,	589	Zacjew, Roman	237
Sherizen, Sanford Ph.D.,	412	Zelkowitz, Marvin V.	132
Sibert, Philip L.	655	Zhang, Cui	103
Smid, Miles E.	514	Zhang, Kui	1
Smith, Dr. Gary	731	Zomback, Robert	358
Smith, Richard E.	247		
Smith, Suzanne T.	370		
Smith, Teresa T.	175		
Smith-Thomas, Barbara	56		
Spalka, Adrian	520		
Spafford, Eugene H.	11, 339		
Sterne, Daniel	254, 400		
Stramella, Anthony	741		
Surface, Tom, LTCOL, USA ...	589		
Swanson, Marianne	581, 734		
Swarup, Vipin	123		
Tajalli, Homayoon	254		
Tipton, Harold	723, 742		

Testing Intrusion Detection Systems: Design Methodologies and Results from an Early Prototype

Nicholas Puketza

Biswanath Mukherjee
Kui Zhang

Ronald A. Olsson

Department of Computer Science
University of California, Davis
Davis, CA 95616

June 23, 1994

Abstract

An Intrusion Detection System (IDS) is a program or set of programs that detects unauthorized uses of computers. As the use of IDSs increases, the need for sound methodologies and tools for testing IDSs is also growing. This paper discusses our development of a software platform that can be used to simulate intrusions in the course of testing IDSs. The platform, which is built on top of the Unix *expect* and *Tcl* packages, allows a user to create their own intrusion scripts. The platform supports the parallel execution of several scripts, so that coordinated intrusions with multiple targets can be simulated. Synchronization mechanisms in the platform are provided so that parallel script tests are reproducible. We are using the platform to create an initial set of test scripts, and we intend to extend this initial script set into a benchmark suite for IDSs. Research in testing methodology for IDSs is guiding the development of our script set. The paper includes results from tests on actual IDSs using our current script set.

Point of Contact: Biswanath Mukherjee
e-mail: mukherje@cs.ucdavis.edu
tel: (916) 752-4826
fax: (916) 752-4767

Testing Intrusion Detection Systems: Design Methodologies and Results from an Early Prototype

Abstract

An Intrusion Detection System (IDS) is a program or set of programs that detects unauthorized uses of computers. As the use of IDSs increases, the need for sound methodologies and tools for testing IDSs is also growing. This paper discusses our development of a software platform that can be used to simulate intrusions in the course of testing IDSs. The platform, which is built on top of the Unix *expect* and *Tcl* packages, allows a user to create their own intrusion scripts. The platform supports the parallel execution of several scripts, so that coordinated intrusions with multiple targets can be simulated. Synchronization mechanisms in the platform are provided so that parallel script tests are reproducible. We are using the platform to create an initial set of test scripts, and we intend to extend this initial script set into a benchmark suite for IDSs. Research in testing methodology for IDSs is guiding the development of our script set. The paper includes results from tests on actual IDSs using our current script set.

Keywords: Intrusion Detection, Testing, Synchronization

1 INTRODUCTION

An intrusion detection system (IDS) is a program or set of programs that attempts to identify individuals who are using a computer system without authorization and those who have legitimate access to the system but are misusing their privileges [Ande90, Mukh94]. Some IDSs monitor a single host, while others monitor several hosts connected by a network. IDSs are now being developed and used at several institutions.

As IDSs are used more and more, the need to develop tools and methods for testing IDSs is growing. This paper discusses our efforts to address that need. Specifically, we are creating a software platform for simulating intrusions. Users of the platform can create their own intrusion scripts and then run them to test the ability of their IDS to detect those intrusions. We are also using this platform to develop an initial set of intrusion scripts that simulate intrusions on Unix systems. The development of the script set is being guided by research into testing methods and concepts. We intend to extend the script set into a benchmark suite that can measure several aspects of the behavior of an IDS while it is running. Finally, we are testing actual IDSs to verify the usefulness of our approaches. For example, we have obtained some quantitative results from "stress-testing" two different IDSs.

To provide some background, Section 2 of this paper discusses the characteristics of IDSs in general. Section 3 describes the software platform development. Section 4 discusses the initial set of test scripts. Section 5 covers testing methods and concepts. Section 6 describes the development of the IDS benchmark suite. Section 7 presents some initial results from testing actual IDSs. Section 8 concludes the paper.

2 INTRUSION DETECTION

The prevalence and ease of use of networking to provide remote access to computers has brought with it a set of previously unanticipated problems. Computer system managers worldwide are extremely concerned with the problem of intrusions, which are unwanted and unauthorized uses of their systems to gain access to (and sometimes to modify) the behavior of their computing resources. These intrusions have often appeared in even the popular news media and have discouraged many organizations from obtaining network connections.

Computer intrusions can range from the subtle (e.g., a user accessing his/her boss' employee reviews) to the extreme as in the actions of the "Wily Hacker" (in Cliff Stoll's book, *The Cuckoo's Egg* [Stol89]), where the purpose of the intrusion was to gain unauthorized access to information on a number of computers in a network.

Examples of intrusions that concern system administrators include:

- unauthorized modifications of system files so as to permit unauthorized access to either system or user information;
- unauthorized access or modification of user files or information;
- unauthorized modifications of tables or other system information in network components (e.g., modifications of router tables in a network to deny use of the network); and
- unauthorized use of computing resources (perhaps through the creation of unauthorized accounts or perhaps through the unauthorized use of existing accounts).

A common element in many of the intrusions of interest is that a single user will often attempt to intrude upon multiple resources in a network. For example, the "Wily Hacker" attempted to use the network to gain access to many sensitive computers on the Internet. Often, detecting the intrusion can be made significantly easier by compiling and integrating evidence of such intrusion attempts across the network rather than attempting to assess the situation from the vantage point of only a single host. For example, an attacker may make only a single attempt at guessing a password for each host computer. Thus, from the vantage point of each of the hosts, each break-in attempt may appear to be a very normal mistake. However, by integrating these observations over the multiple target hosts, it becomes clear that a single attacker is making a concerted attempt to break in somewhere, by looking for an obvious hole.

IDSs attempt to detect the presence of such attacks. Two methods are most often used in IDSs to recognize intrusive behavior. The first method uses expert-system analysis techniques to recognize specific attack command sequences. The second method identifies anomalous behavior by constantly comparing all activity to statistical profiles of normal activity. This method is based on the premise that the behavior of an attacker will be noticeably different from that of a normal user [Denn87].

One approach to computer security is to attempt to create completely secure systems. However, this goal is not feasible in many computing environments. Moreover, it would be impossible to replace the existing huge infra-structure of computer systems that are not completely secure. Intrusion detection provides a practical alternative approach to computer security.

3 A SOFTWARE PLATFORM FOR SIMULATING INTRUSIONS

A basic premise of our work is that an effective way to test an IDS is to simulate intrusions and check if the IDS detects them. We are developing a software platform that facilitates intrusion simulation. Users of the platform can create scripts (similar to Unix shell scripts) that include intrusive commands. The platform provides a script interpreter for running the scripts, which would issue the intrusive commands to the computer system just as if an intruder had typed in the commands.

The software platform consists of extensions to the *expect* [Libes90] package created by Don Libes of NIST. The *expect* package itself is based on the *Tcl* [Oust90] package created by John Ousterhout at U.C. Berkeley.

The *Tcl* (Tool command language) package provides an interpreter for a simple programming language that provides variables, procedures, control constructs such as "if" and "for", arithmetic expressions, lists,

strings and other features. The syntax of *Tcl* is reminiscent of Unix shells, Lisp and C. It is implemented as a C library package. It allows an application to extend the built-in command set with application-specific commands. Recent versions of *Tcl* also support regular expression string specifications. This provides a powerful tool for string pattern matching, which is critical for intrusion detection.

expect is particularly useful for simulating intruders. The *expect* package also provides a programming language interpreter. The core of the *expect* interpreter is the *Tcl* interpreter, but *expect* extends the *Tcl* command set to include several commands for controlling interactive programs. The command "spawn" creates an interactive process (such as *telnet*). The command "expect" waits to receive a specified string pattern (such as "login: ") from the process. The command "send" sends a string to the process. Thus, a script with a sequence of "expect/send" sequences can simulate a human computer user.

A simple script that controls an *rlogin* session would look like the following:

```
#Spawn an rlogin process
spawn rlogin ComputerName -l zhangk
#Expect the password prompt, then send the password.
expect {"Password:" send "ActualPassword \r" }
#Expect the shell prompt, then send commands.
#The shell prompt is specified in a regular expression.
expect {-re ".%|.*>|.##" send "whoami \r" }
expect {-re ".%|.*>|.##" send "ls \r" }
expect {-re ".%|.*>|.##" send "logout \r" }
```

Expect and *Tcl* provide much of the functionality that we need to simulate intrusions. However, we have augmented the *expect/Tcl* core with some additional commands which allow for several scripts running in parallel, including commands for synchronization and communication among the processes that are executing the scripts. The additional commands enable users to simulate more sophisticated intrusions on computer systems. These include intrusions launched by (1) an intruder from multiple terminals (or more likely from multiple windows since window interfaces are ubiquitous) and (2) multiple intruders from multiple terminals.

Synchronization plays a critical role in repeatable testing for scripts that execute in parallel. Often, it may be necessary or desirable to repeat the execution of a test on an IDS, e.g., to learn why (or why not) the IDS failed the test. Repeating the execution of a sequential test (of the single-terminal variety, where the intrusion consists of a single thread of commands) is accomplished simply by re-running the test script on the IDS. Advanced tests (of the multiple-terminal variety, where the intruder may spawn multiple threads of commands), however, may be difficult to repeat because of non-determinism (random latencies) present in the execution of parallel threads. In particular, some of the events in parallel threads may have random execution times; accordingly, the events may not happen in the same order each time the test is run (unless supplemented by some synchronization techniques).

To accommodate "reproducible testing" or "replay" [Hans78, LeBl87], the nondeterministic execution behavior of a test has to be converted to a deterministic form. One solution to this problem is to define a set of a minimum collection of "synchronization sequences" so that the test can provide reproducible results [Tai91]. The synchronization sequences are used to synchronize important events in the parallel threads at strategic locations so that these important events always happen in the same order independent of randomness associated with some events. Our software platform's synchronization mechanisms can be used to create such sequences.

An additional feature of the software platform is a "record and playback" capability. A user can type in a sequence of commands manually, and use the record feature to record that sequence. The "recording" can then be replayed at will, just like other scripts. This feature allows for the quick and easy creation of intrusion scripts. Moreover, it makes the overall platform more useful for those organizations that do not intend to write their own scripts, which requires knowledge of *Tcl* and *expect*.

4 INITIAL SET OF TEST SCRIPTS

Our second major task is to create an initial set of test scripts for Unix-based IDSs. The script set will simulate a variety of known attacks including, for example, those reported by organizations such as CERT and CIAC. We will attempt to classify known attacks, and then represent each class of attacks in the script set. An organization can run the script set and analyze the output of their IDS, checking to make sure that it detects each type of simulated attack. In addition, an organization can supplement our script set with their own private scripts. The script set can be customized to fit a particular computer environment.

Our current scripts simulate the following behaviors:

- browsing through a directory, using the *ls* command to list files, and an editor to view files;
- password-cracking;
- password-guessing using a dictionary file;
- door knob rattling (password-guessing using common passwords);
- attempting to modify system files (e.g., */etc/passwd*);
- “hopping” from computer to computer via *telnet* connections;
- exploiting the *loadmodule* vulnerability.

The script set will be designed to facilitate repetitive, sustained testing. In general, there will be one script for each specific intrusion. Some of these scripts will establish one intrusive connection to a computer, while others will spawn several processes, each of which will then make an intrusive connection. In some cases, concurrently executing processes will coordinate and communicate with each other. Each script will include parameters with variable values, so that the system can be tested over a range of parameter values. A driver script will run the other scripts several times, each time with different values for the various parameters. Thus, the IDS will be exposed to several variations of the same attacks.

5 TESTING METHODOLOGY

The study of testing methodology is guiding the structure of the script set. For example, one testing technique that will be incorporated into our script set is testing in stages. The script set will subject the IDS to three phases of testing: basic functional testing, variance testing, and stress testing. In the basic functional testing phase, each attack can be simulated one at a time. The attack will appear in its most straightforward form. The purpose of this phase is to check if the IDS, in ideal conditions, can detect each attack.

During the variance testing phase, each attack can be simulated several times, and each time some aspect of the attack will be changed. This phase will determine, for example, if there is a variation of an attack that the IDS will not detect, even though it does detect the most straightforward form of the attack.

The stress testing phase will test the IDSs response to the same attacks under extreme conditions. The IDS may not be able cope with a high level of system activity, or perhaps the IDS will be prone to some errors under those conditions. For example, several attacks can be simulated at once, to see if the IDS is still capable of detecting all of them.

Noise (i.e., computer activity that is not part of an intrusive command sequence) will be included in the scripts. The noise level will be varied during the variance-testing and stress-testing phases. Two types of noise that can affect the performance of an IDS will be included in the scripts. Normal background

noise, caused by legitimate user activity, can add stress on the IDS. For example, during periods of heavy LAN traffic, an IDS based on monitoring network traffic must filter more network activity to find possibly suspicious network events. Smokescreen noise, which is activity that is intended to disguise an intrusion, may prevent the IDS from detecting an intrusion. For example, an intruder may pose as a normal programmer and hide the exploratory behavior within the normal programming behavior. A programmer's behavior can be modeled as shown below:

1. with some probability, do an *ls* to check a file;
2. edit a file;
3. compile;
4. with some probability, do an *ls* to check a file;
5. with some probability, go back to step 1;
6. execute the program;
7. with some probability, go back to step 1;

The programming behavior can be used to disguise the *ls* and edit commands which the intruder may be using to examine some target files. Depending on the algorithm that the IDS is using, the IDS may be tricked by this seemingly normal behavior.

5.1 Anomaly Detection Testing

IDSs that use anomaly detection (AD) schemes require some additional considerations with respect to testing. In order to detect anomalous behavior, the AD system usually compares new behavior to statistical profiles of normal behavior. "Training" refers to the development of these profiles of normal behavior. For any AD system, the following considerations can affect system testing:

- What is the training mechanism? (i.e., how are the "normal" profiles created?)
- How much time does training take?
- Is there a way to initialize the AD system with some profiles?
- Does training continue after the system is put into operation?
- Are the profiles *per user* or *per group*?

If the system continues to train itself while it is in operation, then the test scripts should include very gradual behavior changes. This will test if the system can be fooled by an intruder who gradually changes their behavior from "normal" to "intrusive." The test scripts should simulate behavior that produces measurements above and below threshold values in the AD system, to test if the system actually uses the threshold values correctly.

6 Benchmarks

After the initial set of scripts has been created, our next task will be to develop a benchmark suite. The benchmark suite would monitor an IDS while it is running, and measure aspects of the IDS's performance such

as the fraction of known intrusions it can identify, the false alarm rate, and host system resource (memory, CPU load, I/O) consumption. Aspects of this benchmark suite might be modeled after existing benchmark suites such as SPECmarks, Livermore Loops, and Dhrystone, which are used to test the performances of various computer architectures. The information provided by the benchmark suite would help an organization to select an IDS that is suitable for their computer system.

7 INITIAL TEST RESULTS

We have been studying and testing two IDSs: the NSM [Hebe90] (Network Security Monitor) and Haystack [Smah88]. The test results described below are not intended to imply a positive or negative overall assessment of either system. Instead, they are provided to show how the methods and software tools discussed earlier can be used to test real systems.

7.1 NSM Tests

The NSM monitors all the packets that travel in the LAN to which the NSM host computer is connected. The NSM can associate each such packet with the corresponding computer-to-computer connection. It assigns warning values to connections based on the contents of the packets, and on the likelihood of the connection occurring, given a record of recent connections. We ran the NSM on a Sun Sparc Station 2 workstation connected to the Computer Science (CS) LAN segment at UC Davis (UCD). We completed two testing phases: basic functional testing and stress-testing.

7.1.1 NSM Basic Functional Testing

For this phase, we used several different scripts, each designed to simulate a specific intrusive command sequence. The scripts are described in Section 4. Each script would establish a *telnet* connection to another computer, send a sequence of commands to the remote computer, and then close the connection. The NSM monitored the execution of each of these scripts, and assigned a warning value to each connection. For comparison, we also set up the NSM to monitor traffic to and from a busy host in the UCD CS LAN segment for several hours. Although some of this traffic could have been caused by intrusive behavior, we expect that most of it was caused by the legitimate activities of legitimate users.

Ideally, of course, the warning values for connections associated with known intrusive behavior would be high, and warning values for connections associated with normal, benign behavior would be low. Assuming that most of the connections to and from the busy host were normal, the NSM succeeded in this case in assigning a relatively low warning value on average to these connections.

However, the NSM also assigned low warning values to some of the connections associated with the intrusive scripts. We determined, though, that this was caused by *our configuration* of the NSM. Like many IDSs, the NSM can be "tuned" so that it is sensitive to particular intrusive sequences of commands. We are certain that we can now change the configuration of the NSM so that it reports high warning values for all of the intrusive scripts.

Our experience illustrates how basic functional testing can be used to uncover weaknesses in both the IDS itself and the IDS configuration.

7.1.2 NSM Stress Tests

The purpose of the stress tests was to check if the NSM connection reports could be affected by stressful conditions. Stressful conditions for the NSM include all conditions that might cause it to miss packets while monitoring a connection. For example, when the load on the NSM host computer is high, the NSM may be allocated less CPU time, and therefore the probability that it will miss a packet while it waits to run is increased.

We configured the NSM so that it would monitor all connections to and from a specific computer ("computer A") in the LAN. Each test consisted of establishing a certain load on the NSM host, and then running an "intrusion script" on computer A. The script would establish a *telnet* connection from computer A to another computer in the LAN, issue a sequence of several commands to the remote computer, and then close the connection. The intrusion script is a combination of several of the intrusion simulation scripts described previously. To increase the load on the NSM host, a "load script" was run, which simply created a *telnet* connection to the NSM host computer, and then issued an endless sequence of *ps* and *ls* commands. To create higher loads, several processes were created to run the same script simultaneously. The load was measured using the UNIX *uptime* command.

For each test, the NSM produced a report describing the connection established by the intrusion script. Ideally, the report would be identical for each test, because the same script was run for each test. However, we expected the reports to be affected by changes in the load on the NSM host. This was not the case in our first series of stress tests. The connection reports were not affected by increased loads on the host. The NSM is also capable of producing complete transcripts of the connections that it monitors. Inspection of these transcripts confirmed that the NSM had a complete record of the connections and apparently did not miss any packets.

In the next group of tests, we added a second form of stress to the NSM. We used the UNIX *nice* command to lower the "run priority" of the NSM program. Otherwise the tests were the same as the first set of stress tests. However, for this series of tests, the connection reports *were* affected by the increased loads on the host. Apparently, the lowered priority together with a high load on the NSM host caused the NSM to miss some packets. The NSM connection reports include the number of bytes missed for each connection. The NSM calculates this number by monitoring packet sequence numbers. As indicated in Figure 1, the number of bytes missed by the NSM tended to increase as the load on the NSM host increased.

Overall the tests show that the NSM is resilient to stress in the form of host CPU load. In practice, the NSM is probably more likely to be confronted with conditions like those in the first series of stress tests than it is to be confronted with conditions like those in the second series. An attacker may be able to establish several connections to the NSM host, but it is unlikely that the attacker can arrange things so that the NSM process has a low priority compared to the attacker's processes.

7.2 Haystack Tests

Experiments similar to those for the NSM were conducted to perform stress-testing on Haystack. Haystack monitors activities on its host computer by analyzing the system audit trails. The Haystack host for our tests was a Sun Sparc Station 1 workstation with BSM (Basic Security Module) auditing. For each user session, Haystack maintains counts of events such as the number of processes created, the number of files opened, and so forth. These counts are stored in a session record in a database file. Haystack frequently compares these counts against profiles of "normal activity" to detect anomalous sessions. Haystack also includes a misuse detection component which looks for "attack signatures." However, we have been focusing on testing the anomaly detection component.

To put stress on the Haystack system, we adopted the same strategy that we used to test the NSM. Each test consisted of first establishing a certain load on the Haystack host, and then running an intrusion script.

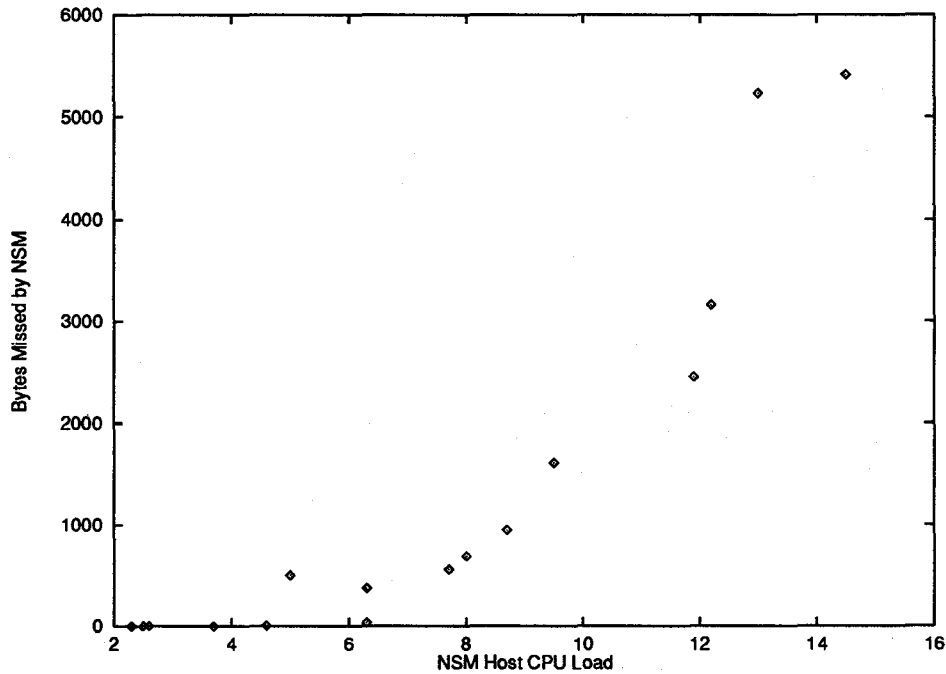


Figure 1: Bytes Missed by NSM vs. NSM Host CPU Load

The script would establish a *telnet* connection to the Haystack host, issue a sequence of several commands, and then close the connection. The intrusion script was nearly the same as the intrusion script used to test the NSM, and the method of increasing the load on the Haystack host was the same as used for the NSM host.

For each test, Haystack would produce a session record describing the connection established by the script. Ideally, the counts in the session record would be identical for each test. Also, Haystack produces six different "suspicion quotients" for each session, which are also stored in the session record. The suspicion quotients indicate to what extent the session exhibits six particular types of behavior associated with intrusions, such as "browsing" and "paranoia." In the ideal case, these scores would also remain the same from test to test, despite the increase in load on the Haystack host.

The test results actually closely matched the ideal case. The session records were all identical, except for two of the counts associated with the number of files opened during the session. These two differences warrant further investigation, but they do not appear to be caused by a system malfunction.

Thus, like the NSM, Haystack proved to be resilient to stress caused by a high load on the host CPU.

7.3 Future Stress Tests

Future stress tests should include requiring the systems to monitor large amounts of activity. Specifically, the NSM should be tested under conditions in which it must monitor several different connections at once, each with a high rate of traffic in terms of packets per second. Similarly, Haystack should be tested in a situation in which several intrusion scripts are running at the same time.

8 CONCLUSION

This work takes several steps toward the systematic testing of IDSs, which we expect will continue to gain importance as IDSs are used more and more. The software tools and testing methods that we are developing can assist not only in the testing of existing IDSs, but also in the development of future IDSs. In addition, we expect to report more rigorous test results in the near future.

References

- [Ande90] J. P. Anderson, "Computer security threat monitoring and surveillance," Technical Report, James P. Anderson Co., April 1990.
- [Denn87] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Eng.*, vol. SE-13, pp. 222-232, Feb. 1987.
- [Hans78] P. B. Hansen, "Reproducible testing of monitors," *Software-Practice and Experience*, vol. 8, pp. 721-729, 1978.
- [Hebe90] L. T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor," *Proc., 1990 Symposium on Research in Security and Privacy*, pp. 296-304, Oakland, CA, May 1990.
- [LeBl87] T. J. LeBlanc and J. M. Mellor-Crummey, "Debugging parallel programs with instant replay," *IEEE Transactions on Computers*, vol. C-36, no. 4, pp. 471-482, April 1987.
- [Libes90] Don Libes, "expect: Curing Those Uncontrollable Fits of Interaction," *Proceedings of the Summer 1990 USENIX Conference*, June 1990.
- [Mukh94] B. Mukherjee, L.T. Heberlein, and K.N. Levitt, "Network Intrusion Detection," *IEEE Network*, May 1994 (to appear).
- [Oust90] John Ousterhout, "tcl(3) - overview of tool command language facilities", unpublished manual page, University of California at Berkeley, January 1990.
- [Smah88] S. E. Smaha, "Haystack: An Intrusion Detection System," *Proc., IEEE Fourth Aerospace Computer Security Applications Conference, Orlando, FL, Dec. 1988*.
- [Stol89] C. Stoll, *The Cuckoo's Egg*, Doubleday, 1989.
- [Tai91] K.-C. Tai, R. H. Carver, and E. E. Obaid, "Debugging concurrent Ada programs by deterministic execution," *IEEE Trans. on Software Eng.*, vol. 17, no. 1, pp. 45-63, Jan. 1991.

A PATTERN MATCHING MODEL FOR MISUSE INTRUSION DETECTION*

Sandeep Kumar

Eugene H. Spafford

The COAST Project
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1398
{kumar,spaf}@cs.purdue.edu

Keywords: intrusion detection, misuse, anomaly.

Abstract

This paper describes a generic model of matching that can be usefully applied to misuse intrusion detection. The model is based on Colored Petri Nets. Guards define the context in which signatures are matched. The notion of start and final states, and paths between them define the set of event sequences matched by the net. Partial order matching can also be specified in this model. The main benefits of the model are its generality, portability and flexibility.

1 Introduction

Computer break-ins are becoming increasingly frequent and their detection is increasingly important. Break-ins make the data residing on computer systems vulnerable to theft and corruption. Compromised sites can also be used to launch further attacks, thus achieving another level of indirection for further break-ins. A majority of break-ins, however, are the result of a small number of known attacks, as evidenced by reports from response teams (e.g. CERT). Automating detection of these attacks should therefore result in the detection of a significant number of break-in attempts.

Intrusion Detection is primarily concerned with the detection of illegal activities and acquisitions of privileges that cannot be detected with information flow and access control models. Examples of these include software engineering flaws in programs that allow cross privilege domain executions, insider abuse and failure of authentication procedures. Intrusion Detection models therefore do not directly overlap with traditional security models [11] which are primarily concerned with modeling information flow in a computer system to ensure that subjects are never able to access unauthorized information, or with modeling access control mechanisms to prevent unauthorized access to objects.

Current approaches to detecting intrusions can be broadly classified into two categories: **Anomaly Detection** and **Misuse Detection** [20]. Anomaly Detection is based on the premise that intrusive activity often manifests itself as an abnormality. The usual approach here is to devise metrics indicative of intrusive activity, and detect statistically large variances on these metrics. Examples might be an unusually high number of network connections within an interval of time, unusually

*This work was funded by the Division of INFOSEC Computer Science, Department of Defense.

high CPU activity, or use of peripheral devices not normally used. This approach has been studied extensively and implemented in a large number of systems [19, 18, 12, 14, 5, 8]. It attempts to quantify the acceptable behavior and thus identify abnormal behavior as intrusive.

The other technique of detecting intrusions, misuse detection, attempts to encode knowledge about attacks as well defined patterns and monitors for the occurrence of these patterns. For example, exploitation of the `fingerd` and `sendmail` bugs used in the Internet Worm attack [21] would be in this category. This technique specifically represents knowledge about unacceptable behavior and attempts to detect its occurrence.

This paper proposes a variation of one approach to misuse detection, state transition analysis, by using pattern matching to detect system attacks. Knowledge about attacks is represented as specialized graphs. These graphs are an adaptation of Colored Petri Nets [9] with guards representing signature context and vertices representing system states. The graph represents the transition of system states along paths that lead to intruded states. Patterns may have user-specifiable actions associated with them that are executed when patterns are matched. The model provides the ability to specify partial orders and subsumes matching of sequences and regular expressions. Patterns also have pre- and post-conditions associated with them that must be satisfied before and after the match. Patterns also may include invariants to specify that a condition is or is not satisfied while the pattern is being matched.

There are several benefits to our approach of using a generic model of matching. A significant benefit is the clean separation of the various components comprising a generic misuse detector. With our approach to designing a generic misuse detector, it can be viewed as three basic abstractions. This enables generic solutions to be substituted for each abstraction without changing the interfaces between the abstractions of the model. These abstractions are:

- The Information Layer. This encapsulates the audit trail and provides a low-level data interface to the monitored computer system.
- The Signature Layer. This provides for a system-independent internal representation of signatures and a system-independent virtual machine to represent the signature context.
- The Matching Engine. This encapsulates the method used to match the patterns. It makes the system independent of any particular choice of matching algorithms. It also allows simple substitution of newer or more powerful mechanisms as they become available.

Furthermore, a standardization of the model of matching signatures permits several external representations of signatures to exist, each facilitating the representation of certain type of signature constructs. Other benefits of the model include its extensibility and portability to different event models; its ability to assign priority to signatures and the ability to dynamically add signatures in the midst of matching [sec. 4].

Our model is generic and does not assume any characteristics of the underlying events against which matching is done or the domain of solution. It provides a mechanism on which matching solutions can be built. For example, the same model applies for the case of monitoring network packet flow, or for monitoring specific patterns in logs generated by general purpose logging utilities. The input events in any of these problem domains can be canonicalized and used as input to our model of matching. Specific instantiations can be made of the model as appropriate to the problem. For example, a specialized version could be created for matching intrusion signatures in the context of UNIX audit trails.

2 Primary Approaches To Misuse Detection

Misuse detection might be implemented by one the following techniques:

1. **Expert Systems**, which code knowledge about attacks as *if-then* implication rules.
2. **Model Based Reasoning Systems**, which combine models of misuse with evidential reasoning to support conclusions about the occurrence of a misuse.
3. **State Transition Analysis**, which represents attacks as a sequence of state transitions of the monitored system [16, 6].
4. **Key Stroke Monitoring**, which uses user key strokes to determine the occurrence of an attack.

These methods are summarized in the following sections.

2.1 Expert Systems

An expert system is defined in [7] as a computing system capable of representing and reasoning about some knowledge-rich domain with a view to solving problems and giving advice. Expert system detectors code knowledge about attacks as *if-then* implication rules. Rules specify the conditions requisite for an attack in their *if* part. When all the conditions on the left side of a rule are satisfied, the actions on the right side of the rule are performed which may trigger the firing of more rules or conclude the occurrence of an intrusion. The main advantage in formulating *if-then* implication rules is the separation of control reasoning from the formulation of the problem solution. Its chief use in misuse detection is to symbolically deduce the occurrence of an intrusion based on the available data.

The primary disadvantage of using expert systems is that working memory elements (the fact base) that match the left sides of productions to determine eligible rules for firing are essentially sequence-less. It is difficult to efficiently specify an order in which to match facts within the natural framework of expert system shells.¹ Other problems include software engineering concerns with the maintenance of the knowledge base [13] and the quality of the rules, which can be only as good as the human devising them [13].

2.2 Model Based Systems

This approach was proposed in [4] and is a variation on misuse intrusion detection. It combines models of misuse with evidential reasoning to support conclusions about its occurrence. There is a database of attack scenarios, where each scenario comprises a sequence of behaviors making up the attack. At any moment the system is considering a subset of these attack scenarios as likely ones being experienced by the system. It seeks to verify them by seeking information in the audit trail to substantiate or refute the attack scenario (*the anticipator*). The anticipator generates the next behavior to be verified in the audit trail, based on the current active models, and passes these behaviors to the *planner*. The planner determines how the hypothesized behavior will show up in the audit data and translates it into a system dependent audit trail match. This mapping from behavior to activity must be easily recognized in the audit trail, and must have a high likelihood of appearing in the behavior.

As evidence for some scenarios accumulates, and decreases for others, the active models list is up-

¹Even though facts are numbered consecutively in current expert system shells, introducing fact numbering constraints within rules to enforce an order makes the Rete match [3] procedure very inefficient.

dated. The evidential reasoning calculus built into the system permits the update of the likelihood of occurrence of the attack scenarios in the active models list.

The advantage of model based intrusion detection is its basis in a mathematically sound theory of reasoning in the presence of uncertainty. The structuring of the *planner* provides independence of representation of the underlying audit trail syntax. Furthermore, this approach has the potential of reducing substantial amounts of processing per audit record. It would do this by monitoring for coarser-grained events in the passive mode and then actively monitoring finer-grained events when those events are detected.

The disadvantage of model based intrusion detection is that it places additional burden on the person creating the intrusion detection models to assign meaningful and accurate evidence numbers to various parts of the graph representing the intrusion model. It is also not clear from the model how behaviors can be compiled efficiently in the planner and the effect this will have on the run time efficiency of the detector. This, however, is not a weakness of the model per se, but a consideration for successful implementation.

2.3 State Transition Analysis

In this approach [16, 6] attacks are represented as a sequence of state transitions of the monitored system. States in the attack pattern correspond to system states and have Boolean assertions associated with them that must be satisfied to transit to that state. Successive states are connected by arcs that represent the events/conditions required for changing state. These conditions, or signature actions, are not limited to a single audit trail event, but may be a complex specification of conditions.

2.4 Keystroke Monitoring

This technique uses user keystrokes to determine the occurrence of an attack. The primary means is to pattern match for specific keystroke sequences indicative of an attack. The disadvantages of this approach are the general unavailability of user typed keystrokes and the myriad ways of expressing the same attack at the keystroke level. Furthermore, without a semantic analysis of the contents, aliases can easily defeat this technique.

2.5 Summary Characterizing These Four Approaches

All four approaches to misuse detection encode and look for specific attacks and use matching in some form to detect them. If an attack is regarded as a set of steps, expert system rules permit the encoding of sequentiality (and other dependencies) between the steps. However, because of the generality of the match procedure of ascertaining firable rules, such dependencies are inefficient to match directly. Model based systems consider 'models' of intrusion and seek to verify them by looking for evidence to corroborate the model. This is done by using matching techniques on the underlying event trail. State transition approaches can be construed as trying to match the sequence of steps that lead a system to a compromised state. Each step in this sequence may, however, require complex computation for determining its occurrence (typically using expert system rules). Key stroke monitoring is the direct application of pattern matching to key stroke logs to match for suspicious or undesirable patterns.

2.6 Benefits And Limitations Of Misuse Detection

A primary disadvantage of anomaly detection, the other major technique for intrusion detection, is that statistical measures of user behavior can be gradually trained. Miscreants who know that they are being monitored can train such systems over a length of time to the point where intrusive behavior is considered normal. Misuse detection is immune to such training: if the signature for an attack is carefully written, even major variations of the same basic attack scenario can be detected. Moreover, the technique is simpler than anomaly detection. Within the framework of misuse signatures, monitoring of system activity can be automated as well.

The primary disadvantage of this approach is that it looks only for *known* vulnerabilities, and is of little use in detecting unknown future intrusions. However, we can look for known patterns of abuse that might occur after a vulnerability is exploited; although the intrusion itself may not be noted, the subsequent actions could be flagged.

3 Intrusion Detection Using Pattern Matching

Our pattern matching is based on the notion of an *event*. Events are auditable changes in the state of the system, or changes in the state of some part of the system. An event can represent a single action by a user or system, or it can represent a series of actions resulting in a single, observable record.

We further specify events as having tags. Generally, monitored events are tagged with data. In particular, the time at which the event occurred is of special importance because of the monotonicity properties of time. The events can have an arbitrary number (though usually a small number) of tag fields. The exact number and nature of the fields is dependent on the type of the event. Mathematically one can think of the events as being tuples with a special field indicating the type of event. For example, one can think of the event a occurring at time t to be the tuple (a, t) , where a denotes the type of the event.

A fundamental requirement of applying pattern matching to intrusion detection is that matching be done with *follows* semantics rather than *immediately follows* semantics. For example, with *follows* semantics the pattern ab specifies the occurrence of the event a **followed** by the occurrence of event b . It does not represent a immediately followed by b with no intervening event. This means that any two adjacent sub patterns within a pattern are implicitly separated by an arbitrary number (possibly zero) of events of any type. This assumption is appropriate in current systems: audit trail generation and modern user interfaces allow users to login simultaneously through several windows thereby generating overlapped entries in the audit trail.

Using *follows* semantics makes the field of discrete approximate pattern matching relevant to intrusion detection. Three characteristics determine the kinds of theoretical bounds that can be placed on the matching solution: 1) whether matching is off-line or online 2) whether signatures can be dynamically added or deleted as matching proceeds and 3) whether all matches of the pattern in the event stream are desired or whether finding a single match is sufficient.

Results in approximately matching various classes of patterns are summarized in fig. 1. These time bounds hold for arbitrary values of deletion, insertion and mismatch costs, and are not optimized for the requirements of misuse intrusion detection. The results are restricted to online matching because we are primarily concerned with real time intrusion detection. *RE* stands for regular expressions and *sequence* refers to a chain of events. The column *match* denotes the type of match determined by the corresponding algorithm. An entry of "all endpts" denotes that the algorithm

Pattern	Time	Space	Preproc	Match	Reference	Comment
Sequence	$O(mn)$	$O(m)$	$O(1)$	all endpts	[22]	Using dynamic programming.
Sequence	$O(mn)$	$O(mn)$	$O(1)$	all	[22]	Using dynamic programming ^a .
Sequence	$O(n)$	$O(m)$	$O(1)$	all endpts	[1, 23]	Pattern fits within a word of the computer. Small integer values of costs.
RE	$O(mn)$	$O(m)$	$O(m)$	all endpts	[15]	Using dynamic programming.
RE	$O(mn)$	$O(mn)$	$O(m)$	all	[15]	Using dynamic programming ^a .

^aDoes not include the time for enumerating all matches, which may be exponential.

Figure 1: Some Results from Pattern Matching Applicable to Misuse Detection

detects all positions in the input where a match with the pattern ends, but cannot reconstruct the match sequence, “all” denotes that the algorithm can also construct the match. Finding all matches of a pattern in the input is an all-paths source-to-sink problem and is computationally hard.

While approximate pattern matching is useful in misuse detection, the general problem cannot be reasonably solved by current pattern matching techniques. For example, it requires matching of partial orders, context-free and context-sensitive structures, and matching in the presence of time, a notion inherent in audit trail generation and very important in specifying intrusions.

After studying common numerous UNIX vulnerability descriptions from such sources as the CERT security advisories, and those detected by the COPS [2] and TIGER [17] tools, we noted a temporally-related partitioning. We were able to classify intrusion attacks on UNIX as follows:

- 1. Existence.** The fact that something(s) ever existed is sufficient to detect the intrusion attempt. Simple existence can often be found by static scanning of the file system. Examples include searching for altered permissions or certain special files.
- 2. Sequence.** The fact that several things happened in strict sequence is sufficient to specify the intrusion.
- 3. Partial order.** Several events are defined in a partial order, for example as in fig. 2.
- 4. Duration.** This requires that something(s) existed or happened for not more than nor less than a certain interval of time.
- 5. Interval.** Things happened an exact (plus or minus clock accuracy) interval apart. This is specified by the conditions that an event occur no earlier and no later than x units of time after another event.

We believe that the vast majority of known intrusion patterns fall into categories 1 and 2. This classification is not strictly a hierarchy as characteristics simple to match at lower levels of the classification become intractable at upper levels. These classes can also be further subdivided into finer categories; details can be found in [10].

3.1 An Overview of Our Model of Matching

We examined various regular methods of representing and matching our attack signatures. Regular expressions can represent only the simplest types of attacks. Context-free and attribute grammars are not easy to extend to approximate matching and do not lend themselves well to a graphical

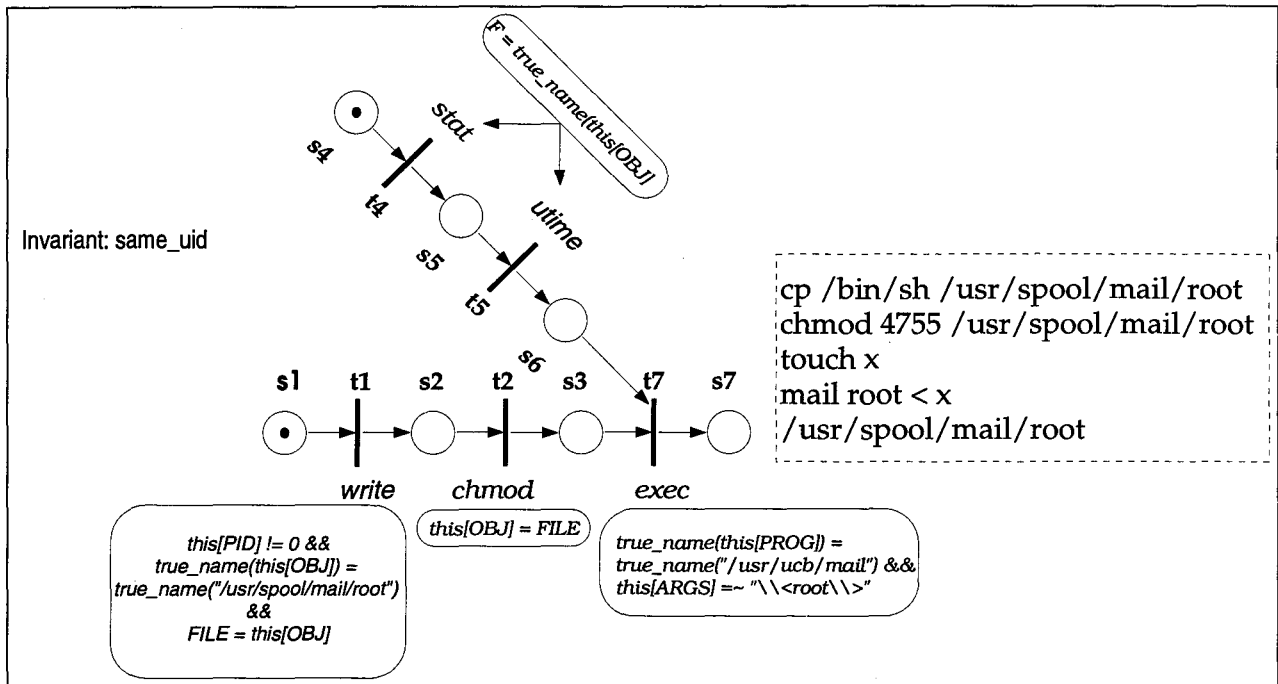


Figure 2: Representing a Partial Order of Events

representation. Regular expressions and context-free grammars do not permit matching to be conditional on the value of specified expressions. Attribute grammars allow conditional matching only in an indirect way. We settled on basing our model of matching on an extension of Colored Petri Nets [9] as they suffer none of these problems.

We refer to each signature represented as an instantiation of a Colored Petri Automaton (CPA). The notion of one or more start states and a unique final state defines the set of strings matched by the CPA. Matching begins with one token in each initial state. The pattern is considered matched for each token that reaches the final state. Along the path to the final state tokens can merge or be duplicated. Partial orders can be written with each trunk of the partial order starting at a different start state. Tokens that are merged carry the merge information with them so the entire merge path is stored.

Patterns are internally stored for matching as CPAs. Externally, a language can be designed to represent signatures in a more programmer-natural framework, and programs in the language compiled to this internal representation. The main differences between our model and CP-Nets are the lack of concurrency in our model, absence of local transition variables, the notion of start and final states, and the notion of pre- and post-conditions and invariants associated with patterns. Moreover, nets in our model are not bipartite, unlike CP-Nets.

Our model is generic and applicable to any well-defined format of input events such as audit trail records, network packets, or other abstractions. Our examples here, however, are taken from the domain of misuse detection in the UNIX environment using audit trails as input.

Consider, as an example, the attack scenario in figure 2 [6]. Its CPA is translated verbatim from the attack scenario for purposes of illustration only. *s1* and *s4* are the initial states of the CPA, and *s7* is its final state. A CPA requires the specification of ≥ 1 initial states (each initial state represents a trunk of the partial order) and exactly one final state. The circles represent states and the thick bars the transitions. At the start of the match, a *token* is placed in each initial state.

Each state may contain an arbitrary number of tokens.

A CPA also has associated with it a set of variables. Assignment to these variables is equivalent to unification. Each token maintains its own local copy of these variables because each token can make its own variable “bindings” as it flows to the final state. In CP-Net terminology, each token is colored, and its color can be thought of as an n -tuple of strings, where the pattern has n variables.

The CPA also contains a set of directed arcs that connect states to other states and transitions. The arcs which connect places to other places are ϵ transitions along which tokens flow nondeterministically without being triggered by an event. Each transition is associated with an event type, called its label, which must occur in the input event stream before the transition will fire. In fig. 2 transition τ_1 is labeled with the event *write*, τ_4 is labeled with the event *stat* and so on. Nondeterminism can be specified by labeling more than one outgoing transition of a state with the same label. There is, however, no concurrency in a CPA: an event can fire at most one transition. A transition is said to be enabled if all its input states contain at least one token.

Optional expressions, or guards, can be placed at transitions. These expressions permit assignment to the CPA variables. Example of these assignments include assignment of values to matched event fields; evaluation of conditions involving equality, $<$, or $>$; and calling built-in and user defined functions. Guards are Boolean expressions which evaluate to *true* or *false*. *this* is a special operator which is instantiated to the most recent event. It may be empty in the case of ϵ transitions. It provides a hook into the event matched at a particular transition. Guards are evaluated in the context of the event which matches the transition label and the set of *consistent* tokens which enable the transition. Tokens are consistent when their variable bindings unify. The set of tokens are unified before being passed to the guards for evaluation.

For example, in order for transition τ_7 to fire, there must be at least 1 token in each of states s_3 and s_6 ; the enabling pair of tokens (one from s_3 , the other from s_6) must have consistently bound (unifiable) pattern variables; and the unified token and the event of type *exec* together must satisfy the guard at τ_7 . A transition fires if it is enabled and an event of the same type as its label occurs that satisfies the guard at the transition. When a transition fires, all the input tokens that have caused the transition to fire are *merged* to one token, and copies of this merged token are placed in each output place of the transition.

The process of merging resolves conflicts in bindings (i.e. makes sure that token bindings unify) between tokens to be merged and stores a complete description of the path that each token traversed in getting to the transition. Thus a token not only represents binding, but also the *composite* path that it encountered on its path to the current state. The sequence of events matched by a CPA is the sequence of events (or partial order) encountered at each transition by the token that has reached the final state.

A CPA is also associated with a pre-condition, a post-condition, and an invariant expression. These are similar to guards that must evaluate *true* to be successful. Patterns that have no transitions (e.g., verifying that root's *.rhosts* file is not world writable) can be specified using pre-conditions to an empty pattern. Post-conditions are provided for symmetry and to allow the recursive invocation of the same pattern.

The reason for having invariants associated with CPAs is more subtle. It seems syntactically inconsistent to us to specify as part of patterns that they *must not* occur while another pattern is being matched. That is, negative pattern specification in a CPA unnecessarily clutters the description of the pattern. The other reason is that the semantics of some invariants cannot be easily absorbed in the framework of transitions and guard expressions. It is more efficient to provide

them as primitives in the matching model than to attempt to subsume them within the framework of matching.

As mentioned earlier, our model of matching is generic. It can easily be instantiated for misuse detection for a system running the UNIX environment, for example. This would involve defining the primitives supported in guard expressions. It might also include coding file test operations, set manipulation functions, system interaction hooks, and other operations. The set of invariant primitives supported in the instantiated model must also be defined. The overall structure of matching remains unchanged.

4 Analysis Of Our Matching Model

There are several difficulties in intrusion detection using pattern matching. The dominant one is the sheer rate at which the data generated by modern processors must be matched. We have some confidence that a system as described in this paper can operate at a speed sufficient to operate in near real time. Furthermore, because state is saved in the tokens and their tag fields, there is no need to save (or re-process) extensive logs of the system.

The other major problem is the nature of the matching itself. An attacker may perform several actions under different user identities, and at different times, ultimately leading to a system compromise. Because an intrusion signature specification, by its nature, requires the possibility of an arbitrary number of intervening events between successive events of the signature, and because we are generally interested in the first (or all) occurrence(s) of the signature, there can be several partial matches of each signature at any given moment. This can require substantial overhead in time and space to track each partial match. In some scenarios, there may be weeks between events. In others, different portions of an attack scenario can be executed over several login sessions and the system is then required to keep track of the partial matches over login sessions. In other cases the signature may specify arbitrary permutations of sub-patterns comprising the pattern thus making the recognition problem much more difficult.

The complexity of matching in our model increases rapidly with increasing complexity of signatures. At the simplest end are patterns without guards, for which algorithms from discrete approximate matching are applicable [fig. 1]. The introduction of guards and variables makes the complexity of the matching problem exponential in the size of the CPA if the description of guards is included in its size. Partial order matching takes super-exponential time. Matching can be improved in some cases by exploiting the monotonic nature of event fields, like the time stamp of the event. Evaluating guards can be optimized by defining a virtual machine for their evaluation. By breaking the guard expressions into sequences of simpler instructions, common subexpression elimination can be performed to reduce the size of the sequence. Such elimination can be done across all the patterns. All these results and optimizations are described in [10].

Our model has several important advantages. It is very portable, in the sense that intrusion signatures can be moved across sites without rewriting to accommodate fine differences in each vendor's implementation. Signatures can also be transparently moved to systems with somewhat different policies and ratings. An abstract audit record definition and a standard definition of a virtual machine to represent guards ensures that patterns pre-compiled to an intermediate representation can be moved across systems with minimal overhead.

Signatures can be dynamically added in the matching engine while maintaining the partial matches of signatures already present in it. The only disadvantage of doing this is that some optimizations, like common subexpression elimination of guards, may not be done for subsequently added patterns

with respect to patterns already compiled in the engine. Actions can also be associated with patterns by incorporating them as expressions in the post conditions.

Signatures can be prioritized by considering each token as a thread of control. Each thread then fetches events from an event manager and acts on them. By prioritizing certain threads, patterns can be prioritized for matching.

5 Conclusions

The paper outlined a pattern matching approach to misuse intrusion detection. It proposed a generic model of matching based on CP-Nets that can be adapted to different problem domains. We used misuse detection using audit trails under UNIX as an example to illustrate the usefulness and applicability of this approach.

The model is interesting and appealing from a theoretical standpoint. However, its true test is an evaluation of its implementation running under "live" conditions. We will implement this model and derive experimental results in the near future. Comparative performance results with other approaches will be difficult because of the lack of standardized benchmarking vulnerabilities and the unavailability of such data for other approaches. We hope that our prototype implementation and benchmarking results will provide the necessary first step in this direction.

References

- [1] R. A. Baeza-Yates and G. H. Gonnet. A New Approach to Text Searching. In *Proceedings of the 12th Annual ACM-SIGIR Conference on Information Retrieval*, pages 168–175, Cambridge, MA, June 1989.
- [2] Daniel Farmer and Eugene H. Spafford. The COPS Security Checker System. In *Proceedings of the Summer Usenix Conference*, pages 165–170, June 1990.
- [3] Charles L. Forgy. RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. In *Artificial Intelligence*, volume 19. 1982.
- [4] T. D. Garvey and T. F. Lunt. Model based Intrusion Detection. In *Proceedings of the 14th National Computer Security Conference*, pages 372–385, October 1991.
- [5] L. T. Heberlein, K. N. Levitt, and B. Mukherjee. A Method To Detect Intrusive Activity in a Networked Environment. In *Proceedings of the 14th National Computer Security Conference*, pages 362–371, October 1991.
- [6] Koral Ilgun. USTAT: A Real-Time Intrusion Detection System for UNIX. Master's thesis, Computer Science Department, University of California, Santa Barbara, July 1992.
- [7] Peter Jackson. *Introduction to Expert Systems*. International Computer Science Series. Addison Wesley, 1986.
- [8] R. Jagannathan, Teresa Lunt, Debra Anderson, Chris Dodd, Fred Gilham, Caveh Jalali, Hal Javitz, Peter Neumann, Ann Tamaru, and Alfonso Valdes. System Design Document: Next-Generation Intrusion Detection Expert System (NIDES). Technical Report A007/A008/A009/A011/A012/A014, SRI International, March 1993.
- [9] Kurt Jensen. *Coloured Petri Nets – Basic Concepts I*. Springer Verlag, 1992.

- [10] Sandeep Kumar and Eugene Spafford. An Application of Pattern Matching in Intrusion Detection. Technical Report 94-013, Purdue University, Department of Computer Sciences, March 1994.
- [11] Carl E. Landwehr. Formal Models for Computer Security. *ACM Computing Surveys*, 13(3):247–278, September 1981.
- [12] G. E. Liepins and H. S. Vaccaro. Anomaly Detection: Purpose and Framework. In *Proceedings of the 12th National Computer Security Conference*, pages 495–504, October 1989.
- [13] Teresa F Lunt. A Survey of Intrusion Detection Techniques. *Computers & Security*, 12(4):405–418, June 1993.
- [14] Teresa F. Lunt, R. Jagannathan, Rosanna Lee, Alan Whitehurst, and Sherry Listgarten. Knowledge based Intrusion Detection. In *Proceedings of the Annual AI Systems in Government Conference*, Washington, DC, March 1989.
- [15] Eugene W. Myers and Webb Miller. Approximate Matching of Regular Expressions. In *Bull. Math. Biol.*, volume 51, pages 5–37, 1989.
- [16] Phillip A. Porras and Richard A. Kemmerer. Penetration State Transition Analysis – A Rule-Based Intrusion Detection Approach. In *Eighth Annual Computer Security Applications Conference*, pages 220–229. IEEE Computer Society press, IEEE Computer Society press, November 30 – December 4 1992.
- [17] David R. Safford, Douglas L. Schales, and David K. Hess. The TAMU security package: An outgoing response to internet intruders in an academic environment. In *Proceedings of the Fourth USENIX Security Symposium*. USENIX Association, 1993.
- [18] M. Sebring, E. Shellhouse, M. Hanna, and R. Whitehurst. Expert Systems in Intrusion Detection: A Case Study. In *Proceedings of the 11th National Computer Security Conference*, October 1988.
- [19] Stephen E. Smaha. Haystack: An Intrusion Detection System. In *Fourth Aerospace Computer Security Applications Conference*, pages 37–44, Tracor Applied Science Inc., Austin, TX, Dec 1988.
- [20] Stephen E. Smaha. Tools For Misuse Detection. In *Proceedings of ISSA '93*, Crystal City, VA, April 1993.
- [21] Eugene Spafford. The Internet Worm Report. Technical Report 823, Purdue University, February 1990.
- [22] Robert A. Wagner and Michael J. Fischer. The String-to-String Correction Problem. In *Journal of the ACM*, volume 21, pages 168–178, january 1974.
- [23] Sun Wu and Udi Manber. Fast Text Searching With Errors. Technical Report TR 91-11, University of Arizona, Department of Computer Science, 1991.

Artificial Intelligence and Intrusion Detection: Current and Future Directions

Jeremy Frank

frank@cs.ucdavis.edu

Division of Computer Science
University of California at Davis
Davis, CA. 95616
(916) 752-2149
NSA URP MDA904-93-C-4085
ARPA DOD/DABT63-93-C-0045

June 9, 1994

Abstract

Intrusion Detection systems (IDSs) have previously been built by hand. These systems have difficulty successfully classifying intruders, and require a significant amount of computational overhead making it difficult to create robust real-time IDS systems. Artificial Intelligence techniques can reduce the human effort required to build these systems and can improve their performance. Learning and induction are used to improve the performance of search problems, while clustering has been used for data analysis and reduction. AI has recently been used in Intrusion Detection (ID) for anomaly detection, data reduction and induction, or discovery, of rules explaining audit data. We survey uses of artificial intelligence methods in ID, and present an example using feature selection to improve the classification of network connections. The network connection classification problem is related to ID since intruders can create "private" communications services undetectable by normal means. We also explore some areas where AI techniques may further improve IDSs.

Keywords: Artificial Intelligence, Intrusion Detection, Feature Selection.

1 Problems in Intrusion Detection

Intrusion Detection (ID) is the identification of attempted or ongoing attacks on a computer system or network. Issues in ID research include data collection, data reduction, behavior classification, reporting and response. Although there are many significant open problems in ID research, we focus on data reduction and classification. *Data reduction* consists of analyzing a collection of data in order to identify the most important components of the data, thereby reducing processing time, communications overhead and storage requirements. *Classification* is the process of identifying attackers and intruders. Artificial intelligence (AI) techniques have been used in many IDSs to perform these important tasks.

Section 2 of this paper will briefly discuss artificial intelligence methods and describe some of the methods which will appear in this paper. Section 3 will discuss the problem of data reduction and discuss how AI methods have been used in a variety of IDSs to solve this problem. Section 4

will discuss the application of AI to the classification problem. Section 5 will present an example of the use of feature selection to improve the classification of network connections, and section 6 will discuss some future applications of AI in IDSs.

2 Artificial Intelligence Methods

Artificial Intelligence is concerned with improving algorithms by employing problem solving techniques used by human beings. Humans excel at tasks such as *learning*, or gaining the ability to perform tasks from examples and training. An *expert system* handles problems using a computer model of expert human reasoning. However, most expert systems must undergo continuous maintenance to perform well [WeKu].

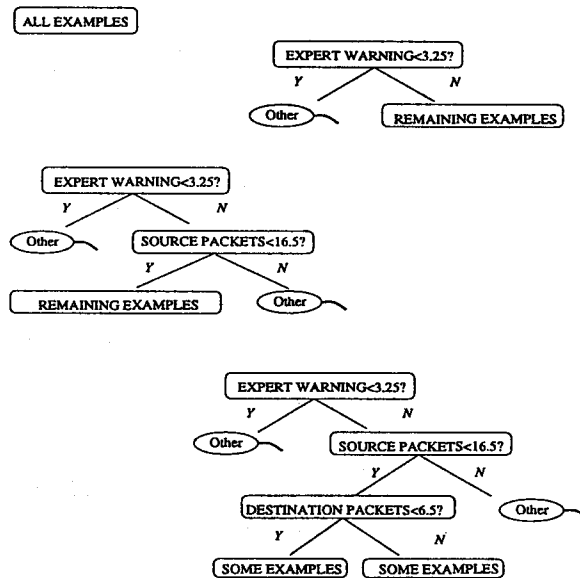


Figure 1: Growing a Decision Tree.

Other systems can acquire knowledge from a set of *training instances*. These training instances can be questions and correct answer pairs, or problems and the steps of a solution. *Rule Based Induction* derives rules which explain the training instances more clearly than a mathematical or statistical analysis of data [WeKu]. *Classifier systems* attempt to learn how to classify future examples from a set of training data. An example of a system that can be used as a classifier is a *Neural Network*, which uses a model of biological systems to perform classification. Neural networks are characterized by highly connected networks which are trained on a set of data in the hopes that the network will correctly classify future examples [WeKu]. Another example of a classifier is a *Decision Tree* [BuCa] [WeKu]. Decision trees are constructed by finding ways to separate the data into two or more groups. We then separate each of these groups in turn, until we have small groups of examples left. Decision tree algorithms are designed to find the best questions to ask so that most or all of the examples in each group belong to one class. Figure 1 shows how a tree used to classify network connections is constructed. The goal of *Feature Selection* is to reduce the amount of information required to make good predictions, and to improve the error rate of classifiers [WeKu]. This is accomplished by searching subsets of features, or information sources, and testing the ability of those features to classify the training instances. The search process itself

is the subject of continuing research in the AI community. Humans are also able to generalize or abstract from large amounts of information by a process called *discovery* or *clustering*. *Data clustering* techniques are used to group data together according to some criteria [ShDi]. Clustering is used to discover hidden patterns in data that humans might miss.

3 Data Reduction for Intrusion Detection

Due to the massive amount of audit data available, classification by hand is impossible. For example, a user typically generates between 3-35Mbytes of data in an eight hour period and it can take hours to analyze a single hour's worth of data. Analysis is difficult even with computer assistance because extraneous features can make it harder to detect suspicious behavior patterns. Complex relationships exist between the features which are difficult for humans to discover. IDSs must therefore reduce the amount of data to be processed. This is especially important if real-time detection is desired. Therefore, some form of data reduction is required for IDSs. Reduction can occur in one of several ways. Data that is not considered useful can be filtered, leaving only the potentially interesting data. Data can be grouped or clustered to reveal hidden patterns; by storing the characteristics of the clusters instead of the data, overhead can be reduced. Finally, some data sources can be eliminated using feature selection.

3.1 Data Filtering

The purpose behind data filtering is to reduce the amount of data directly handled by the IDS. Some data may not be useful to the IDS and thus can be eliminated before processing. This has the advantage of decreasing storage requirements and reducing processing time. However, filtering may throw out useful data, and so must be done carefully.

In systems such as DIDS [SnBr], MIDAS [SeSh], TIM [TeCh] and NSM [HeDi], data filtering is done using heuristic or ad hoc methods, which can be viewed as expert rules for filtering. Other systems filter data in a more adaptive or dynamic way. [DeBe] present a filtering system based on a neural network which acts to filter data which does not fit an observed trend. They assume that user activity contains notable trends that can be detected, and that there are correlations among the collected audit data. Regularity ensures that the network will pick up the regular trends exhibited, and automatically account for correlations in the input data. Using a type of neural network called a *recurrent network* ensures that behavior trends can be accurately recalled. The network "forgets" behavior over time, and can thus adjust to new trends. Thus the network acts as a filter to determine whether or not an audit record fits the regular trends.

3.2 Feature Selection

In complex classification domains, some data may hinder the classification process. Features may contain false correlations which hinder the process of detecting intrusions. Further, some features may be redundant since the information they add is contained in other features. Extra features can increase computation time, and can impact the accuracy of an IDS. Feature selection improves classification by searching for the subset of features which best classifies the training data [SiSk]. In the ID domain, features are derived from information sources used to detect intrusions, and training instances are derived from detected intrusion attempts as well as normal behavior. Thus, feature selection can be used to find features most indicative of misuse, or can be used to distinguish between types of misuse. [Do] and [SiSk] have performed comparisons of a variety of

feature selection techniques, and [Do] tested several techniques on simulated computer attack data to explore the possibility of using feature selection to improve intrusion detection techniques. In section 5 we give an example of feature selection applied to classifying network connections.

3.3 Data Clustering

Clustering can be performed to find hidden patterns in data and significant features for use in detection. Clustering can also be used as a reduction technique by storing the characteristics of the clusters instead of the actual data. Artificial Intelligence researchers have noted the close relationship between learning and data compression. Discovering the generalization of a concept is in essence finding a more compact representation of the set of objects, and a hierarchical clustering algorithm can be used for inductive generalization [Th]. Statistical clustering measures the probability that each example is in a given cluster. Exemplar methods build a representative of each cluster throughout the clustering process. Distance clustering uses a distance measure to establish membership in a cluster. Conceptual clustering requires that an object meet necessary and sufficient conditions for cluster membership.

PRAD [LaBe] uses k -nearest neighbor (knn) clustering to reduce data. To perform knn clustering, x percentiles of the distribution are determined. The data is reduced to one of the values 1 to x . Thus each of n data elements is clustered with $k = \frac{n}{x+1}$ neighboring data points. Along with the choice of number of percentile points, the positions of the percentiles can also be located. For instance, [La] uses 2 percentiles and splits the categories at the 50th percentile. The Bernoulli vector used in Haystack uses $x = 2$ and splits at the 90th percentile. [He]

Wisdom&Sense [LiVa] also performs clustering of numerical data. The history of audit data is separated into clusters which correspond to high density regions followed by low density regions; the historical data is then represented by clusters which represent each density region.

4 Behavior Classification in Intrusion Detection

Classifying user or system behavior is a very hard problem. One problem is that only a small fraction of behavior is misuse; another is that often misuse looks like normal use, so it can be difficult to distinguish between intruders and normal users. As a result, classification can result in "false negatives", wherein an attacker is misclassified as a normal user. "False positives" can also degrade productivity in the systems being protected by invoking countermeasures unnecessarily. Finally, all types of intrusive behavior can't be identified in advance. Several AI techniques have been used to improve IDS classification performance. Statistical anomaly detection works on the assumption that many attackers behave differently from normal users, or that a system or process behaves differently during an attack. If a user is behaving abnormally it may indicate an attacker using that user's account. Expert systems encode policy statements and known attacks as a fixed set of rules. User behavior is matched to these rules to determine if an attack is under way. Rule-based systems create (discover) and manage rules corresponding to anomalous behavior.

4.1 Expert Systems

In an expert system, a set of rules encoding knowledge of an "expert" are used to make conclusions about information gathered by the IDS. The rule set must be modified by hand, and may incorporate a statistical or probabilistic component. However, in specialized domains expert systems can outperform humans.

IDES contains a rule-based component which encodes knowledge about past intrusions, known system vulnerabilities, and security policy. IDES rules are encoded in an expert system shell. As information is gathered, the expert determines whether or not any rules have been satisfied, then chooses the most appropriate rule to select [LuJa]. [DeBe] propose an expert system in connection with a neural network. The neural network component reports anomalies to the expert system, which also employs data not used by the net. The expert contains a rule base similar to that used in IDES, with known attacks and system policy information. It also provides the network with contextual inputs that audit data does not provide, and ensures that the network does not train on intrusive behavior. Other IDSs employing expert systems are Haystack [He], AudES [Ts], MIDAS [SeSh] and NADIR [JaDu].

4.2 Anomaly Detection

Anomaly detection is based on the assumption that misuse or intrusive behavior deviates from normal system use [LuTa] [De] [DeBe] [LiVa]. In many cases this is a valid assumption, as in the attacker who breaks into a legitimate user's account. The attacker may behave differently than the regular user, so if the IDS has established what the user normally does during a session, it can determine that the user is not behaving normally and detect the attack. IDSs constructed with this philosophy learn profiles of behavior and report anomalies to either a human or another part of the IDS for more detailed analysis. An anomaly detection system contains three distinct phases:

1. Abstract local information
2. Evolve background information from local abstractions
3. Establish anomaly background boundaries.

[Ma] discusses smoothing raw data to eliminate reliance on outlying data points, blending data using an exponential method to weight historical data higher than current data, and finding and blending the variation in behavior to establish a tolerance level for network anomaly detection. Anomaly detection can be difficult since the concept of normal can change over time. Furthermore, normalcy can be established with respect to different time frames. For example, a system can establish session, daily and weekly trends.

PRAD [La] learns profiles of resource usage, time information, and directory access patterns. Profiles are analyzed with respect to login sessions and time windows, and performance in windows is weighted over time. Windows extend across logins so that information across login sessions can be maintained in the profiles. PRAD provides a means for including changing legitimate user behavior in profiles after legitimacy is ascertained. Other systems employing statistical anomaly detection are MIDAS[SeSh] , IDES [JaVa], NADIR [JaDu] and Haystack [He].

4.3 Rule-Based Induction

In contrast to expert systems, rule-based systems automatically develop rules to explain the historical data they collect. Rules are modified over the lifetime of a system in order to keep the rule set accurate and manageable.

In Wisdom&Sense, rules are generated which specify legal values of features conditioned on the values of other features. Legality is determined from the history of data for each feature. Rules can overlap in specificity due to incomplete information in the history. Rule pruning occurs if there are too many legal values for a feature, too few historical values, the rule is too deep, if rules overlap, or

a rule is conditioned on a previously (in the forest of rules) determined anomalous value. All rules can either be used to signal anomalies, or the most appropriate rules to use may be determined [LiVa]. TIM's rules remain in the rule base only if they are highly predictive or confirmed by many observations. Prediction is calculated using an entropy model. The user must specify the behavior TIM is trying to predict. Rules are stored in a lattice, and predict an outcome with a specific accuracy based on the observed audit history. Both short term and long term patterns are checked against for anomalies. Rules also support instantiation. TIM allows the user to enter rules describing either patterns or abstractions of the audit data. [TeCh]

5 An Experiment Using Feature Selection

A growing problem in intrusion detection is network-based intrusion detection. Since computer systems are increasingly network dependent it is imperative to protect both local and regional networks. An example of the kinds of problems that must be faced can be seen in the problem of classifying a network connection. On UNIX systems a connection is characterized by the source port and destination port numbers. Certain ports are reserved for different services; e.g. telnet uses port 23. However, an intruder can hide network connections by strategically placing the servers that receive the connections on different ports[He2]. The mapping of ports to services is internal to a single machine; an intruder could also change the port map. Thus we would like to be able to identify the type of connection made without referring to port numbers.

We examined how feature selection can improve classification of network connections by minimizing the classification error rate and by reducing the number of features required to classify connections. To do so, we analyzed three feature selection algorithms to test methods for selecting the best subset of features to classify connections using decision trees. We conducted two types of experiments: one selecting features which distinguish one type of connection from all others, and one which classified all connection types. Our data consisted of 15,947 connections from one local area network during one week of normal use.

5.1 NSM Features

We collected information about network connections using NSM [HeDi]. NSM returns data about each connection. We collected the following information for each connection:

Feature	Feature type
Index	int
Expert system warning	float
Time in seconds	int
# of packets from source	int
# of packets from destination	int
# of data bytes from source	int
# of data bytes from destination	int

Most of the fields are self-explanatory; the expert system warning value is NSM's expert analysis of how likely the connection is to be an attack. Each piece of information collected was used as a feature for our experiments. In addition to these fields we also collected the actual connection type for use in training.

5.2 Search Algorithms

To reiterate, feature selection is used to reduce the amount of information required to make good predictions, and to improve the error rate of classifiers. Since our task is to correctly classify future examples based on the training examples, we used the classification error rate of a decision tree to evaluate each set of features [BuCa] [WeKu]. This error rate is computed by counting the misclassifications on a test data set which is independent from the training set [BuCa]. We tested representative algorithms from each of 3 broad classes of search algorithms.

Backward sequential search begins with the full set of features. At each stage of the search, each feature in the remaining set is removed. The best feature to eliminate from the set is determined by comparing the error rates of the classifiers created using the resulting feature sets. Backward Sequential Search runs in polynomial time [Do].

Beam search is a type of best-first search which uses a bounded queue to limit the scope of the search. The queue is ordered from best-state to worst-state, with the best state placed in the front of the queue. The algorithm operates by taking the first state in the queue, the most promising state, and extending the search from that state as in Backward Sequential Search. Each new state visited is placed in the queue in order of the goodness of it's state. If there is no limit on the length of the queue, then Beam Search takes exponential time to complete but if the queue length is 1 then Beam Search takes polynomial time. Thus accuracy can be increased if the queue length is increased [Do].

In Random Generation Plus Sequential Selection, we perform several sequential selections from different places in the search space. As mentioned, the goal is to avoid picking the first good feature set seen on the assumption that other good feature sets are also available. To do so, we generate a random feature set, then perform backward and forward sequential selection on the state. Random Generation runs in polynomial time but is more expensive than Backwards Sequential Selection [Do].

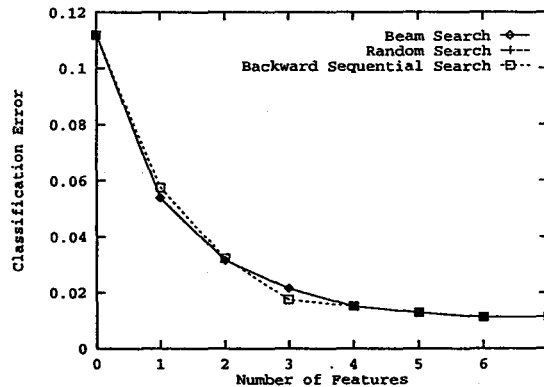


Figure 2: Feature Set Size vs Error Rate For Classifying Connections.

5.3 Algorithm Performance

The table below shows the size, classification error rates, and number of states searched for each problem. Note that with 7 features the total number of subsets of these features is 128. We can see here that in most cases using a computationally expensive algorithm did not gain much in terms of numbers of features or error rate with the exception of the shell classification problem.

Further, the RGSS algorithm always repeated states in its search. This implies that differentiating connections can be done reasonably well with a less-than-exhaustive search. However, the fact that the random search did find a better classifier indicates that sophisticated algorithms can be worth the cost. Notice that the classification error is smaller than the percentage of the smallest class of connections observed.

Problem	Algorithm	Number of States	Size of Best Feature Set	Error Rate
All	Beam	53	6	0.011266%
All	BSS	29	6	0.011266%
All	RGSS	206	6	0.011266%
SMTP	Beam	53	5	0.007231%
SMTP	BSS	29	5	0.007231%
SMTP	RGSS	190	5	0.007231%
Login	Beam	38	4	0.001177%
Login	BSS	29	4	0.001177%
Login	RGSS	188	4	0.001177%
Shell	Beam	38	4	0.002018%
Shell	BSS	29	4	0.002018%
Shell	RGSS	178	4	0.001009%

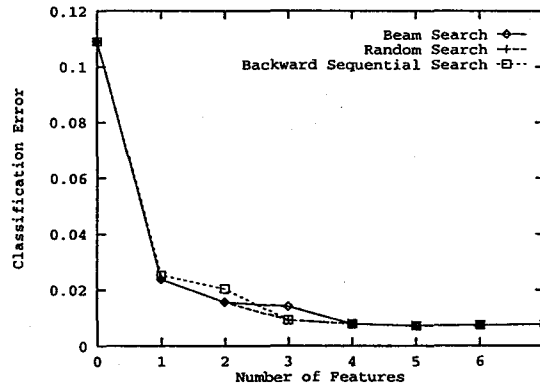


Figure 3: Feature Set Size vs Error Rate For Identifying SMTP Connections.

5.4 Error Rate Performance

The following graphs show how error rate varied with size of feature sets found for each of the problems. We note that in all cases the algorithms performed similarly. We also note that in all cases except the shell connections problem each algorithm found not only the same sized best feature set but that the sets were composed of the same best features.

Figure 2 shows the error rate vs feature set size for classifying all network connections. The error rates are for the best feature set of the indicated size found by each algorithm. We see that with as few as 3 features the error rate is < 0.02%. However, the best feature set only excluded the number of destination data bytes. The 3 features found by all of the search algorithms were time in seconds, packets from the destination and source data bytes, and had an error of 0.017488%. Thus for this particular case a sophisticated algorithm wasn't necessary to find a good classifier.

Figure 3 shows the error rate vs feature set size for classifying SMTP connections. We do not see a jump in the error rate until 3 features, although the best feature set contained 5 features, with the Index and Destination Data bytes excluded. All algorithms found that using time in seconds,

packets from the destination and source data bytes gave an error of 0.009248%, so again the less computationally expensive algorithm is the best choice.

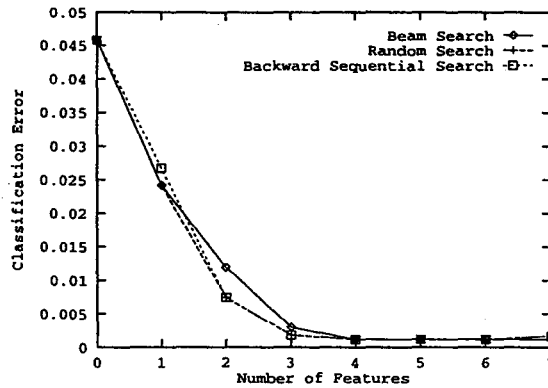


Figure 4: Feature Set Size vs Error Rate For Identifying Login Connections.

Figure 4 shows the error rate vs feature set size for classifying login connections. Again we do not see a jump in the error rate until 3 features, although the best feature set contained 4 features. This time the index, destination data bytes and source data bytes were excluded from the best feature set. Again BSS and RGSS performed the best, with RGSS finding that time in seconds, packets from the destination and source data bytes gave an error of 0.001850%. Again BSS found the same feature set and expanded fewer states.

The results from classifying shell connections were more interesting. While all 3 algorithms found the best feature set size was 4, RGSS found a better feature set with a better error rate, as shown below.

Algorithm	Index	Expert	Time	Src Pkts	Dest Pkts	Src Data	Dest Data
Beam	N	Y	N	Y	Y	Y	N
BSS	N	Y	N	Y	Y	Y	N
RGSS	N	Y	Y	Y	N	Y	N

Here, all algorithms found sets of 3 features with 5 times the error rate of the best feature set.

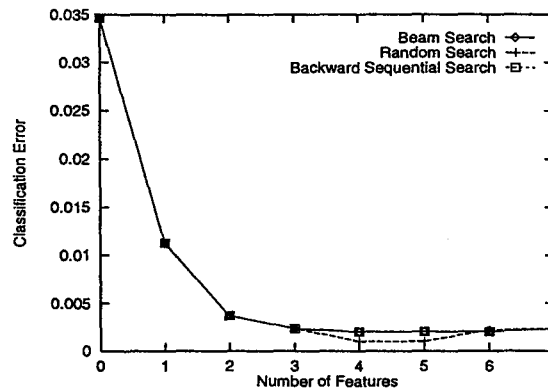


Figure 5: Feature Set Size vs Error Rate For Identifying Shell Connections.

Figure 5 shoes shows the error rate vs feature set size for classifying shell connections.

6 Future uses of AI in Intrusion Detection

Currently, many IDSs employ AI methods in their systems. We expect AI techniques will improve understanding of how non-intrusive and intrusive behavior differ, as well as enable hierarchical classification of different types of attacks.

6.1 Feature Selection in Intrusion Detection Systems

We have demonstrated that feature selection can be effective in a small example. We can extend feature selection to intrusion detection. NSM returns more data than we used in our experiment, and can also be executed containing a list of strings to look for in connections. This information can be invaluable in determining whether critical files or commands pass across the network. However, the number of interesting strings to check for can number easily in the hundreds. Another method of deriving new features is by analyzing multiple transcripts and combining information for a single source-destination pair. Since an attack might span multiple “attempts” over time, a single source-destination pair might appear many times with different data each time. Aggregating statistics and adding others can result in more features. For instance, the number of times a particular connection was made is such an aggregated feature.

6.2 Reconfiguration and Customization of IDSs

IDSs can be site specific. Using data reduction techniques we can customize an IDS to a particular site by finding the information sources most useful to that site’s IDS needs. We can also re-configure an IDS using feature selection after finding new data sources. For instance, NSM can be configured to search for different strings on a network. Feature selection can be used to determine which strings are the best to search for.

IDSs such as DIDS, COPS, Haystack and IDES all make assumptions about the type of data they collect. Feature selection techniques can be modified to analyze the value of the features used in other IDSs and perhaps to enhance their performance by eliminating noisy features. Systems like DIDS and IDES may be difficult to analyze, since they incorporate classifier systems already. In some cases the classifiers may rely on all audit features being present before making decisions; if those features are not present it may cause them to incorrectly classify (or fail to classify) behavior.

6.3 Clustering in Intrusion Detection

We envision that clustering will be very useful in intrusion detection. We plan to use clustering techniques such as Autoclass [ChKe] to explore patterns in audit and network data. One way clustering can be used in ID is by giving an overview of complex data. Another is by considering each cluster in turn and analyzing interesting characteristics of each cluster. For instance, if one cluster has characteristics of two other extremely dis-similar clusters, it may indicate usage patterns midway between two “normal” groups, leading to suspicions of misuse.

7 Conclusions

We have provided a brief survey of AI methods used in a variety of IDSs. We have also demonstrated how one technique, feature selection, can be used to reduce overhead and improve classification of

network connections. Other IDSs already make extensive use of AI techniques to improve their ability to detect attacks on computer systems.

8 Acknowledgements

Todd Heberlein provided valuable assistance with NSM for this paper. Justin Doak's work provided theoretical basis for this work, and we used his code to provide the data available. He also provided valuable technical assistance. Becky Bace of the NSA also provided valuable comments on the draft of this paper. This work was supported by NSA under University Research Program under Contract Number MDA904-93-C-4085 and by ARPA under Contract Number DOD/DABT63-93-C-0045.

References

- [Ba] R. Baldwin. "Kuang: Rule-Based Security Checking." *COPS documentation*, MIT, Lab For Computer Science Programming Systems Research Group, 1989.
- [BuCa] W. Buntine, R. Caruna. "Introduction to IND and Recursive Partitioning." *IND Documentation*, NASA Ames Research Center, 1991.
- [ChKe] P. Cheeseman, J. Kelley, M. Self, J. Stutz, W. Taylor, D. Freeman. "Autoclass: A Bayesian Classification System." *Proceedings of the 5th International Conference on Machine Learning*, 1988.
- [De] D. Denning. "An Intrusion Detection Model." *IEEE Transactions on Software Engineering*, vol SE-13, no.2, 1987.
- [DeBe] H. Debar, M Becker, D. Siboni. "A Neural Network Component for an Intrusion Detection System." *Proceedings, IEEE Symposium on Research in Computer Security and Privacy*, 1992.
- [Do] J. Doak. "Intrusion Detection: The Application of Feature Selection, A Comparison of Algorithms, and the Application of a Wide Area Network Analyzer." PhD. Thesis, Department of Computer Science, University of California, Davis, 1992
- [He] T. Heberlein. "Haystack's Analysis: A Brief Description." Internal Document, University of California, Davis, 1991.
- [He2] T. Heberlein, Private Communication, March 1994.
- [HeDi] L. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J Wood, D. Wolber. "A Network Security Monitor." *Proceedings, IEEE Symposium on Research in Computer Security and Privacy*, 1990.
- [JaDu] K. Jackson, D. DuBois, C. Stallings. "An Expert System Application for Network Intrusion Detection." 1991
- [JaVa] H. Javitz, A. Valdes. "The SRI IDES Statistical Anomaly Detector." *Proceedings, IEEE Symposium on Research in Computer Security and Privacy*, 1991.
- [LaBe] L. Lankewicz, M. Benard. "Real-time Anomaly Detection Using a Non-Parametric Pattern Recognition Approach." *Proceedings 7th Annual Computer Security Applications Conference*, 1991.

- [La] L. Lankewicz. "A Non-Parametric Pattern Recognition to Anomaly Detection." PhD. Thesis, Dept. of Computer Science, Tulane University, 1992.
- [LiVa] G. Liepens, H. Vacaro. "Anomaly Detection: Purpose and Framework." *Proceedings, 12th National Computer Security Conference*, 1989.
- [LuJa] T. Lunt, R. Jaganathan, R. Lee, A. Whitehurst, S. Listgarten. "Knowledge-Based Intrusion Detection." *Proceedings of the AI Systems in Government Conference*, 1989.
- [LuTa] T. Lunt, A. Tamaru, F. Gilham, R. Jaganathan, P. Neuman, C. Jalali. "IDES: A Progress Report." *Proceedings of the Sixth Annual Computer Security Applications Conference*, 1990.
- [Ma] R. Maxion, "Anomaly Detection for Diagnosis." *FCTS*, 1990
- [RiSc] C. Riesbeck, R. Schank. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, New Jersey, 1989.
- [SeSh] M. Sebring, E. Shellhouse, M. Hanna, R. Whitehurst. "Expert Systems in Intrusion Detection: A Case Study." *Proceedings of the 11th National Computer Security Conference*, 1988.
- [ShDi] J. Shavlik, T. Dietterich. *Readings in Machine Learning*. Morgan Kauffman, California, 1990.
- [SiSk] W. Siedlecki, J. Sklansky. "On Automatic Feature Selection." *International Journal of Artificial Intelligence*, vol.2, no.2, 1988.
- [SnBr] S. Snapp, J. Bretano, G. Dias, T. Goan, L. Heberlein, C. Ho, K. Levitt, B. Mukherjee, S. Smaha, T. Grance, D. Teal, D. Mansur. "DIDS: Motivation, Architecture and an Early Prototype." *Proceedings of the 14th National Computer Security Conference*, 1991.
- [TeCh] H. Teng, K. Chen, S. Lu. "Adaptive Real-Time Anomaly Detection Using Inductively Generated Sequential Patterns." *Proceedings, IEEE Symposium on Research in Computer Security and Privacy*, 1990.
- [Th] C. Thornton, "A Computational Model for the Data Compression Metaphor." *Proceedings of the 3d International Conference on Artificial Intelligence*, 1990.
- [Ts] G. Tsudik. "AudES - An Expert System for Security Auditing." *Proceedings of AAAI Conference on Innovative Applications in AI*, 1988
- [WeKu] S. Weiss, C Kulikowski. *Computer Systems That Learn*. Morgan Kauffman, California, 1991.

A THREE TIER ARCHITECTURE FOR ROLE-BASED ACCESS CONTROL

Ravi S. Sandhu and Hal Feinstein*

SETA Corporation
6858 Old Dominion Road, Suite 200
McLean, VA 22101

This paper presents a reference architecture (or conceptual framework) for the specification and enforcement of role-based access control (RBAC). The architecture has three tiers in loose analogy to the well-known ANSI/SPARC architecture for database systems. (Although we take our inspiration from the database domain, we emphasize that our proposed RBAC architecture is germane to applications and systems in general and is not limited to databases per se.) The three tiers of the reference architecture consist of (i) multiple external or user views concerned with the utilization of RBAC in a specific context within the organization, (ii) a single conceptual or community view which amalgamates diverse external views into a consistent and unified composite suitable for overall security administration, and (iii) multiple internal or implementation views concerned with enforcement of RBAC in various subsystems of the enterprise information system. This paper discusses these three tiers and their interrelationships. We demonstrate the usefulness of this conceptual approach, and identify issues which need further research to make this framework a reality.

1 INTRODUCTION

Role-based access control (RBAC) is an idea whose time has come. A consensus has developed in recent years that the traditional discretionary and mandatory access controls (DAC and MAC, respectively) embodied in DoD's landmark Orange Book [Dep85] are inadequate for the information security needs of many commercial and civilian Government organizations (as well as single-level military systems, for that matter). Orange Book DAC is too weak for effective control of information assets, whereas Orange Book MAC is focused on US policy for confidentiality of classified information. RBAC has therefore emerged as a third form of access control to fill this urgent need.

Although RBAC is perceived to be a good match for the information security needs of a wide spectrum of organizations (which are not being currently served by Orange Book DAC and MAC) there remains a lack of consensus about exactly what RBAC means. For example, participants at the recent Federal Criteria Workshop felt that while "RBACs were needed in the commercial/civilian sector," at the same time "roles are a new concept and not yet well understood" [Nat93b].

It is beyond the scope of this paper to give a complete definition of RBAC, let alone one on which wide consensus has been achieved. Such an attempt would be premature. Our purpose here is to present a conceptual framework, or reference architecture, for specifying and enforcing RBAC. Our framework has three tiers in loose analogy to the well-known ANSI/SPARC architecture for database systems [Te78]. Although we take our inspiration from the database domain, we emphasize that our proposed RBAC architecture is germane to applications and systems in general and is not limited to databases per se. Much as the ANSI/SPARC framework is useful independent of the particular data model employed, our proposed RBAC framework is useful whatever the final consensus definition of RBAC turns out to be.

*Ravi Sandhu is also affiliated with the Department of Information and Software Systems Engineering at George Mason University in Fairfax, VA.

Our reference architecture is motivated by two main considerations. Firstly, a number of proposals incorporating one form or another of RBAC have been published in recent years. Some of these have been incorporated in commercial products, and more such products can be expected to appear in the near future. Vendors tend to integrate RBAC facilities in products in different ways, because of the economics of integrating such features into existing product lines. Over time the emergence of standards will impose some order in this arena, but the near term is likely to display a divergence of approaches. Even as standards emerge, we can expect a diversity of support for RBAC due to the longevity of legacy systems.

Secondly, in large organizations there will be a large number of roles and complex relationships between the roles and permissions authorized by them. In most contexts it would be appropriate to take a simplified view appropriate for the task at hand. For example, in some situations all members of a particular department can be treated as belonging to a single role; whereas in other situations more refined roles such as managers, technical staff and administrative staff need to be distinguished.

The central tier of our architecture resides in a single community view of RBAC as it applies to the entire organization in question. This community view will typically be large and complex reflecting the reality of modern organizations. The specialized context-specific views of RBAC tailored to particular applications and situations are accommodated in multiple user views that reside above the central tier. The views of RBAC embodied in different products are embodied in multiple implementation views residing below the implementation tier. Figure 2 illustrates these three tiers. The central tier serves as the focal point for mapping the external user views to the internal implementation views.

The rest of this paper is organized as follows. Section 2 briefly reviews prior work on RBAC. Section 3 presents our three-tiered reference architecture for RBAC. Section 4 discusses issues pertaining to the all important central tier. Sections 5 and 6 respectively discuss relationships between the top two tiers and the bottom two tiers of our architecture. Section 7 gives our conclusions.

2 BACKGROUND

The roots of RBAC can be traced back to the earliest access control systems. RBAC has a superficial resemblance to the long-standing use of user groups in access control systems. There are, however, two very important differences between groups and roles; as articulated by Ferraiolo and Kuhn [FK92].

Firstly, groups are essentially a discretionary mechanism whereas roles are non-discretionary. The ability to assign permissions to a group is usually discretionary (although the authority to assign members to a group is usually non-discretionary, and reserved for the security administrator). Thus, the owner of a file can decide what access a particular group has to that file. On the other hand, the allocation of permissions to a role, as well as determination of membership in a role, are both intended to be non-discretionary.* In the simplest case, these decisions are made solely by the security administrator. More generally, the security administrator can selectively delegate this authority to other users or roles in the system (as recognized in the CS-3 profile of the Draft Federal Criteria [Nat92]).

Secondly, the nature of permissions allocated to a role is significantly different than the usual read, write, execute, etc., supported by typical Operating Systems (OSs). Ferraiolo and Kuhn define

*Not all proposals for RBAC agree with this position. For example, relations in Oracle [Ora92] can be owned by individuals who have discretionary authority regarding how to assign permissions for these relations to users and roles. In our opinion the non-discretionary aspect of roles is very important. In systems such as Oracle, it is possible to achieve a de facto non-discretionary behavior by strict control of ownership of relations which contain corporate data.

the notion of a transaction as a program (or transformation procedure) plus a set of associated data items. The operation authorized is therefore to execute the specified program on this set of data items. This very important notion allows authorization in terms of abstract operations embodied in transformation procedures. For example, the bank teller role can be allocated the authorization to execute credit and debit operations on accounts rather than to general read and write operations. This enables RBAC to address security for applications in terms of the application's operations, as opposed to generic read and write operations in a general-purpose OS.

Roles have been employed in several mainstream access control products of the 1970s and 80s, such as IBM's RACF and Computer Associates' CA-ACF2 and CA-TOP SECRET. These products typically include roles for administrative purposes. For example, RACF provides an Operator role with access to all resources but no ability to change access permissions, a Special role with ability to change permissions but no access to resources, and an Auditor role with access to audit trails (including events generated by Operator and Special, who have no access to the audit trail) [Mur93]. The use of roles for administrative purposes also appears in context of cryptographic modules [Nat93a]. Here User, Crypto-Officer and Maintenance roles are distinguished.

Recent proposals for RBAC, such as Ferraiolo and Kuhn [FK92], go beyond this traditional use of roles by providing them at the application level to control access to application data. This is an important innovation which makes RBAC a service to be used by applications. RBAC offers the opportunity to realize benefits in securing an organization's information assets, similar to the benefits of employing databases instead of files as the data repository. Instead of scattering security in application code, RBAC will consolidate security in a unified service which can be better managed while providing the flexibility and customization required by individual applications. It should be noted that access control similar to RBAC has often been embedded in application code. The point is to move this functionality out of application code into a common set of services.

Over the past five years or so, several proposals for RBAC have been published. Some of these, such as [Bal90, Ste92, Tho91], have proposed extensions to existing access control systems to incorporate roles. Commercial products, such as ORACLE [Ora92], have incorporated roles. Roles are also being considered as part of the emerging SQL3 standard [PB93]. Proposals for incorporating roles in object-oriented systems have been published [LW88, Tin88]. More recently Ferraiolo and Kuhn [FK92] of NIST have given an abstract and unifying description of the essential characteristics of RBAC. Their ideas have been incorporated in the CS-3 protection profile of the Draft Federal Criteria [Nat92]. The application of roles for enforcing static and dynamic separation of duties has also been recognized [CW87, San88b, San91].

The formulations of RBAC mentioned above have been motivated by different considerations. Not surprisingly they differ in important aspects. At present there is no unified model with respect to which these different formulations can be viewed as special cases. Development of such a model, and a taxonomy of its special cases, would be a significant contribution to this area. This task is beyond the scope of this paper. Our concerns here are independent of the unified RBAC model that may eventually emerge.

3 THE THREE TIER FRAMEWORK

In the late 1970s, an ANSI/SPARC study group published a report [Te78] which has had an enduring impact on database systems. This report described a three-tier "architecture" for a database, consisting of:

1. the external or user view which is concerned with the way data is viewed by end users,
2. the conceptual or community view which amalgamates diverse external views into a consistent and unified composite, and

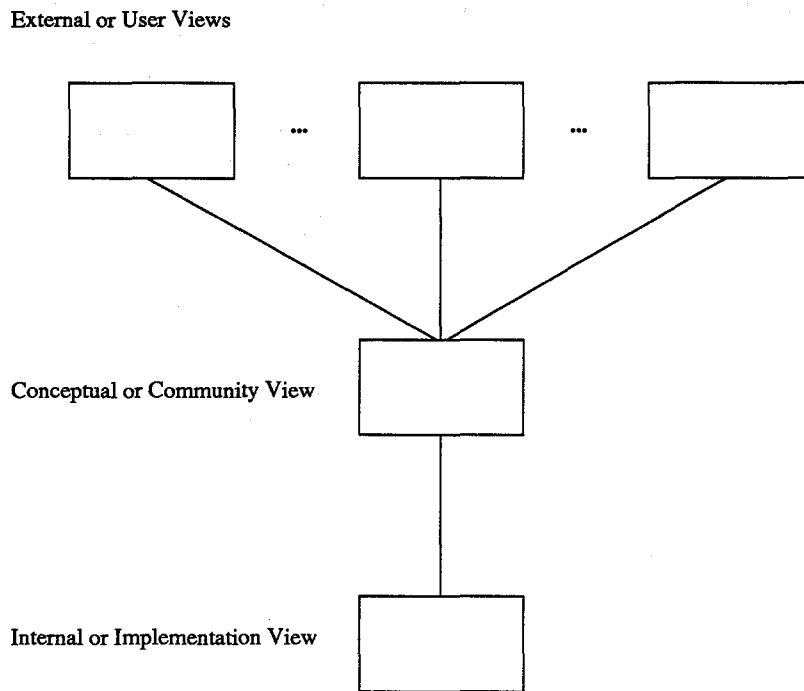


Figure 1: ANSI/SPARC Database Architecture

3. the internal or implementation view which is concerned with the way that data is actually stored.

This database architecture is shown in figure 1.

Note that there are multiple external views, but only a single conceptual and a single internal view. This three-tier approach to database systems has stood the test of time, and is remarkably independent of the particular data model being used.

We believe a similar approach is suitable for developing a common framework or reference architecture for RBAC. RBAC is concerned with the meaning and control of access control data (i.e., data used to control access to the actual data of the organization). In other words we are concerned with a special purpose database system. It is therefore sensible to adapt the approach used for general-purpose database systems. However, there is one significant difference. In database systems, it is intended that the implementation will eventually be on a particular database management platform. Consequently, the internal or implementation view is closely tied to the particular platform that is selected. With RBAC we do not have the luxury of assuming a homogeneous implementation environment. Instead we must confront the reality of heterogeneous implementations up front. This leads us to modify the three-tier ANSI/SPARC architecture by introducing multiple internal views, corresponding to different platforms on which the implementation is done. This RBAC reference architecture is shown in figure 2.

Our three-tiered approach to RBAC therefore consists of

1. multiple external views,
2. a single conceptual view, and

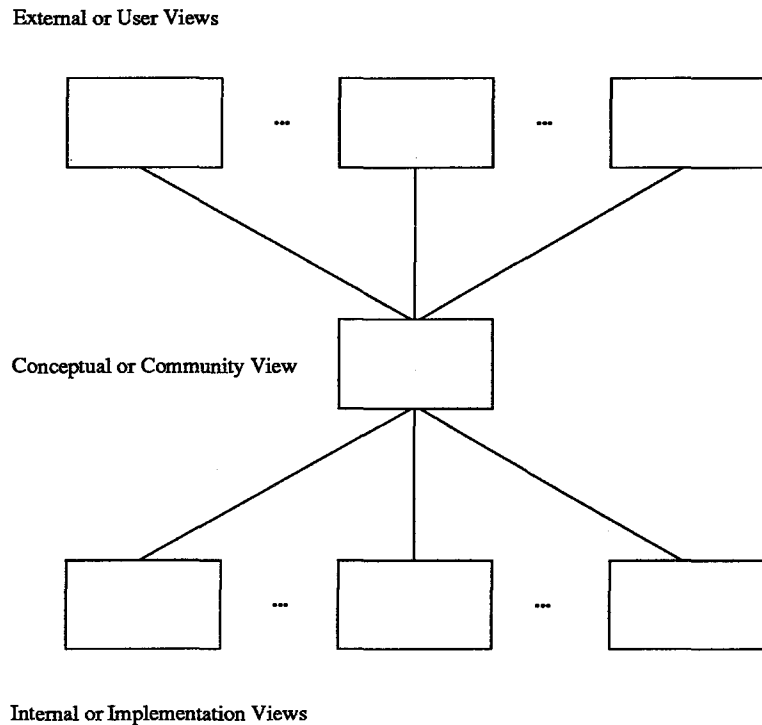


Figure 2: A Three Tier Architecture for RBAC

3. multiple implementation views

Next, let us consider the appropriate model for each of these tiers. We again turn to the ANSI/SPARC architecture for inspiration. There is a conspicuous difference between the models used at the implementation and conceptual tiers. We expect a similar difference in our RBAC reference architecture. Why is this so? We expect the model used at the conceptual level to have richer constructs and primitives, because it is intended to express a composite system-wide view of RBAC. Practical considerations will inevitably dictate that not all these features can be directly supported in an implementation. Hence the implementation models will be simpler and less user-friendly. Moreover, we expect a range of sophistication from rather primitive mechanisms (say on a vanilla UNIX platform) at one end to very elaborate ones (say on an object-oriented database management system) at the other. Note that this viewpoint lets us accommodate legacy systems co-existing with newer ones. It should also be clear that the effort required to translate a conceptual view will be less or greater depending upon the sophistication of the implementation platform being targeted. In some cases, a translation may not even be feasible (or practical) without enhancement of the target platform.

The difference between the conceptual and external tiers is less marked. Whether or not there should be any difference is open to debate. For relational databases, both tiers are often identical and directly based on the relational data model. However, sometimes a richer model such as the entity-relationship model is used for the external view while a relational model is used at the conceptual view. We anticipate a similar situation in the RBAC reference architecture. Based on the historical experience with the ANSI/SPARC architecture, it might well happen that initially the same RBAC model is used at both tiers, but over time richer models are developed for the external view.

In subsequent section we first discuss the all important central tier of our RBAC reference architecture. This is followed by discussion regarding the top two tiers and their relationship. Finally we discuss the relationship between the bottom two tiers.

4 THE CENTRAL TIER

The central tier of our reference architecture consists of a single community view of RBAC applicable to the entire information system and its myriad applications. This community view is the essential conceptual vehicle for effective deployment of enterprise-wide RBAC. Development of a suitable model of RBAC for this tier is an all important task, but beyond the scope of this paper. Here we discuss some issues in constructing such a model, and describe some desirable characteristics that this model should have. We should say at the outset that an RBAC model suitable for this tier must be rigorous, have a formal foundation and yet be intuitively comprehensible and useful to practitioners. This is a daunting task, but one which we feel can be accomplished in future work.

RBAC is intended to be a flexible and customizable vehicle for application security. A recent NIST study [FGL93] of "current and future information technology security needs of the commercial, civil, and military sectors" concluded that, "Each organization viewed its access control needs as unique. Access control mechanisms need to be applied on a case-by-case basis in meeting individual computer security threats." Ferraiolo and Kuhn [FK92] similarly state that, "A wide gamut of security policies and needs exist within civilian government and private organizations. An organizational meaning of security cannot be presupposed."

In order to achieve flexibility, it is important to resist imposing a particular form of RBAC in all situations. A general RBAC model must instead accommodate a variety of alternatives that can be selected on a case by case basis. At the same time it is not very useful to enumerate a long menu of alternatives as a general RBAC model. The goal of flexibility and customization must be reconciled with the need for simplicity and minimality of concepts in the model.

To illustrate this conflict, consider the question of whether or not a user can simultaneously take on more than one role. In many situations it can be argued that limiting the user to one role at any time is beneficial for purpose of least privilege. For example, the role of being an employee and the role of being a stockholder of an enterprise are two independent attributes of a user yielding different access rights which should be separately exercised by an individual. Similarly, the role of being a physician and being a patient should be regarded as mutually exclusive. At the same time, there are many situations when it is beneficial to let a user exercise multiple roles simultaneously. This is particularly so when the roles are based on competence or skill. Thus an attorney can take on the role of a specialist in, say, tax and criminal law. A single individual, cleared to both specialist roles, can then be assigned to a case requiring both kinds of attorneys. A system which insists on users taking on only one role at a time would require two individuals to process the case, or perhaps a single individual who is required to switch back and forth between roles. Requiring two individuals, where one would do, is clearly inefficient. Requiring frequent switching back and forth of roles, in this situation, is the sort of thing that gets users frustrated with security.

This example demonstrates that any system which enforces one alternative to the exclusion of the other, is going to be awkward to use when the situation at hand does not match the alternative hardwired in the system. One approach to resolving the particular conflict of this example is to recognize two kinds of roles: those that can be simultaneously held and those that cannot. More generally, one could imagine disjoint sets of roles which can be mixed under permissible combinations specified in some formal language. If one is not careful, this kind of thinking can lead to models which are extremely general and open-ended, and thereby lose their value. Rather than customizing such a general model, it may be more useful for the practitioner to construct a model more directly applicable to the need at hand.

The issue of how many and which roles can be simultaneously exercised by a user, is but one of many such issues which need to be addressed in constructing a RBAC model. Perhaps, the most fundamental issue that needs to be addressed is what exactly is meant by a role. Ferraiolo and Kuhn [FK92] define a role as follows: "A role can be thought of as a set of transactions that a user or set of users can perform within the context of an organization. Transactions are allocated to roles by a system administrator. . . Membership in a role is also granted and revoked by a system administrator."

The question then arises as to what is a transaction. Ferraiolo and Kuhn provide two definitions. In the first definition, a transaction is defined as "a transformation procedure, plus a set of data items accessed by the transformation procedure." In this case Ferraiolo and Kuhn observe that access control is very simple, because it "does not require any checks on the user's rights to access a data object, or on the transformation procedure's right to access a data item, since the data accesses are built into the transaction." All that needs to be checked is whether or not the user is authorized to run the transaction in question (via some role). Ferraiolo and Kuhn also offer a more sophisticated definition of transaction by redefining it to "refer only to the transformation procedure, without including a binding to objects." Access control enforcement must then check 5-tuples of the form (u,r,t,o,x) to ascertain whether or not a user u in role r can access object o in mode x using transaction t (x is one of read, write, append, etc.). The need for such fine-grained access control has been supported by comments from an IRS representative at the NIST-NSA Federal Criteria Workshop [Nat93b, page 47].

So even the definition of a transaction has several important variations. The foregoing aspect concerns the nature of privileges that are associated with roles. There is also significant variation concerning the manner by which privileges and users are assigned to roles. On one hand this assignment should be non-discretionary, and perhaps done only by the system administrator. On the other hand, in large systems this will be an onerous responsibility to impose on a single individual or office. To facilitate security administration it should be possible for the system administrator to delegate pieces of this authority to other users or roles. The need for such delegation is recognized in the Commercial Security profiles of the draft Federal Criteria [Nat92], as well as in [FK92]. There is, however, a great deal of variation in how this delegation can be accomplished, especially if delegation of such administrative privileges can be further delegated. For example, the manager of a department may be given some administrative control over roles pertaining to that department. However, we would like to impose some non-discretionary controls on the manager so that, for example, the manager can delegate his authority to certain roles but not others. A general RBAC model must allow variation here without stipulating the universal use of one approach.

Another important aspect of RBAC in which there is significant variation concerns inheritance of privileges across roles. In general, roles can be composed of other roles [FK92]. To take an example from [FK92], the Intern role can be assigned to the Healer role, so that members of the Intern role automatically obtain membership in the Healer role. Taking this one step further, a Doctor role can be assigned to the Intern role. In this manner members of the Doctor role become members of the Intern role, and transitively members of the Healer role. There are significant policy issues that arise in this context. In this particular example transitive propagation of membership appears to be justified. On the other hand, transitive propagation may not always be desirable. It may also be useful to distinguish the privileges of a role that may be inherited through other roles, from privileges that are private to a role and cannot be inherited. In a truly general model we may also wish to consider denials (or negative privileges), in addition to permissions (or positive privileges). This is a useful facility, particularly when there are multiple administrative authorities in a system. The exact semantics of inheritance of privileges in such cases can become extremely murky [Lun88]. It is also important to develop systematic methodologies for designing and maintaining such hierarchies of roles. The techniques described in [San88a] for construction of such hierarchies in the context of protection groups could be employed here.

The point of the preceding discussion is that there are many variations to be considered in a model for RBAC. The dimensions that were considered above are summarized below.

- What, and how many, roles can a user exercise simultaneously?
- What is the granularity of privileges that can be assigned to roles?
- How do privileges granted to roles interact with privileges granted to users as individual?
- How is security administration of assignment of users and privileges to roles accomplished?
- How are privileges inherited when roles are composed of other roles?

We need a common approach towards modeling RBAC wherein variations along the dimensions identified above (and possibly others which emerge).

Let us now consider the nature of an RBAC model suitable for the central tier of the RBAC reference architecture. In abstract terms any access control model has to address the following issues.

1. What is a protection state?
2. What does it mean?
3. How is it changed?

To illustrate this let us consider some classical access control models, and see how they address these issues. The most widely used model to date is perhaps the Bell-LaPadula or BLP model [BL75]. In BLP a protection state consists of a set of subjects SUB, a set of objects OBJ, a discretionary access matrix D, a current access matrix M, and a function SECURITY-LEVEL which maps each subject and object to a label from the given security lattice. The meaning of the protection state is that M specifies which accesses are currently authorized. The D and M components of the protection state are changed in different ways. D is changed at the discretion of subjects. A subject who owns an object controls access to that object. M is changed when an access is actually attempted. If D authorizes the access, and the simple-security and star-properties permit it, the relevant right is entered in the appropriate cell of M. Only the security officer can change the sets SUB and OBJ.

For another example of how these three issues are addressed consider the typed access matrix (TAM) model of Sandhu [San92] (which is obtained by adding strong typing of subjects and objects to the classical HRU model [HRU76]). In TAM a protection state consists of a set of subjects SUB, a set of objects OBJ, an access matrix AM whose cells contain entries from a set of rights R and a function TYPE which maps each subject and object to a type in the specified set of types. The meaning of the protection state is that AM specifies which accesses are currently authorized, as well how AM can be currently modified. The sets SUB and OBJ, and the access matrix AM are changed by executing one of a given collection of commands. A command will execute only if AM authorizes its execution.

A conceptual RBAC model will follow this established paradigm. Each of the three issues identified above need to be formally defined in some appropriate notation. Some of the components of such a model have been identified by Ferraiolo and Kuhn [FK92]. There is, however, much that remains to be done. As we have argued there are many variations regarding the precise behavior of RBAC. A vital component of this subtask is the development of a common framework that can accommodate these variations. In practice many of the "advanced" features of RBAC may not be needed in all applications, and may not be supported in all products. Nevertheless a complete RBAC model must address the problem in its full generality. Restricted versions of the model can then be identified as needed. This approach is consistent with the ordered ranking of protection profiles in

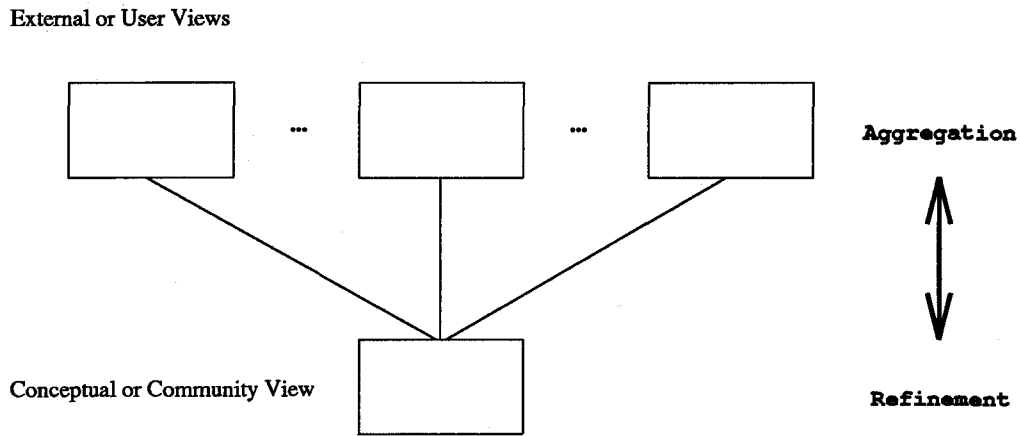


Figure 3: Harmonizing the Top Two Tiers

the draft Federal Criteria [Nat92]. It is important to analyze which aspects of RBAC add significant expressive power, and which are just dispensable conveniences.

In conclusion the RBAC model for the central tier must be a flexible and general model. It should be rigorously defined, have a solid formal foundation and yet be intuitively comprehensible and useful to practitioners. Although this is a challenging task, we feel it can be accomplished relatively soon.

5 HARMONIZING THE TOP TWO TIERS

Let us now consider the relationship between the top two tiers of the reference architecture, reproduced in figure 3. Each external view gives one perspective on the common community view, relevant to the particular context at hand. The relationship between the the top two tiers is one of aggregation and refinement as indicated in the figure.

Aggregation is a process by which several distinct roles are combined into a single role, because the distinction is not relevant in the given context. For example, the community view might have distinct roles for, say, Undergraduate Students, Master's Students and Doctoral Students. In an application where all students are treated alike, these roles could be collapsed (i.e., aggregated) into a single Student role. In other applications, which confer different privileges to the various student roles, this distinction is significant. Refinement is simply the opposite operation to aggregation.

Different external views will aggregate different collections of roles from the community view. Some external views may aggregate the student roles into a single one. Others may keep the distinction between student roles but aggregate distinct faculty roles into one. Still others may aggregate both or none of the student and faculty roles. Our expectation is that a relatively small portion of the overall role set from the community view will be needed more or less intact in a particular external view. Most of the roles will, however, be aggregated. In other words each external view will see only a small part of the roles set in all its detail.

So long as entire roles are being aggregated or refined, the mapping between the top two tiers is relatively simple. There may be situations where the role relevant to the external view does not come about so cleanly by aggregation. For example, suppose the community view has roles *A* and *B*, whereas the external view requires a role which has some (but not all) members of *A* and some

(but not all) members of B . We identify below some techniques for accommodating such an external view.

- One could modify the community view to create a new role C and explicitly assign those members of A and B who should belong to this role. This treats A , B and C as unrelated roles.
- One could modify the community view to partition A into A_1 and A_2 (with $A_1 \cap A_2 = \phi$), and B into B_1 and B_2 (with $B_1 \cap B_2 = \phi$) so that $C = A_1 \cup A_2$ can be defined in the desired external view. This would require external views which use A to now treat A as an aggregate of A_1 and A_2 , instead of being a role form the community view. Similarly, for external views which use role B .
- We could allow aggregation which can select the appropriate subsets of A and B , based on some condition for identifying members who should belong to the aggregated role C . This will complicate the aggregation operation and might dilute the central role of the conceptual view.

This list is not intended to be exhaustive. The point is that various alternatives are available as the community and external views adapt to the ever changing demands of the applications. One needs a systematic methodology for dealing with such changes.

6 HARMONIZING THE BOTTOM TWO TIERS

Now consider harmonization of the bottom two tiers, shown in figure 4. Each of the implementation views will aggregate roles from the community view. The aggregation done here will constrain which external views can be hosted on which implementation views. An implementation view that aggregates distinct student roles into a single role obviously cannot support an external view that requires this distinction to be maintained. In an ideal situation the implementation view may do no aggregation, in which case it could support all the external views. In practice, however, one would expect considerable aggregation to occur; if only because of legacy systems which have directly built in the external view without consideration of the common community view. Performance considerations may also require such aggregation to occur. Note that in both figures 3 and 4 aggregation is in the direction away from the central community view, and refinement is directed towards this view.

The second mapping shown in figure 4 is between implicit and explicit mechanisms. This mapping recognizes that the implementation platform may not support all the features of RBAC in the community view. For example, role hierarchies may not be supported. Suppose there are two roles Faculty and Staff such that every member of the Faculty role is automatically a member of the Staff role (but not vice versa). Thus a new faculty member need only be enrolled in the Faculty role, and will automatically be enrolled in the Staff role. This facility is often called role inheritance in the literature. Support for role inheritance in the community view is highly desirable, but such support will not be available on every implementation platform. To continue our example, at the community view it suffices to enroll a new faculty member into the Faculty role. However, in the implementation view the new faculty member will need to be enrolled in both Faculty and Staff roles. Similarly, a departing faculty member needs to be removed from the Faculty role in the community view; but in the implementation view requires removal from both Faculty and Staff roles.

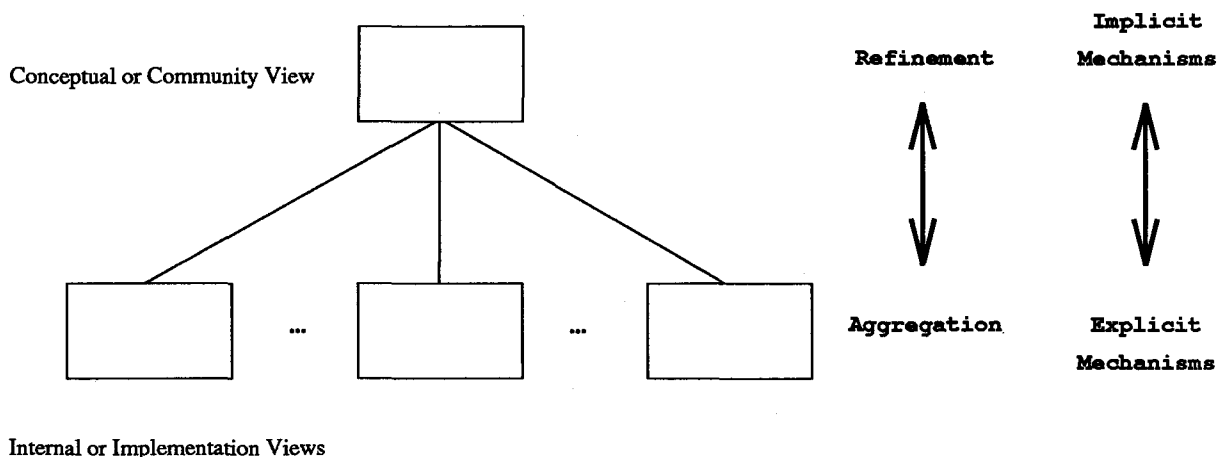


Figure 4: Harmonizing the Bottom Two Tiers

7 CONCLUSION

In this paper we have proposed a three-tiered reference architecture for role-based access control (RBAC), and have identified some of the issues that need to be addressed in making this framework a reality. Our reference architecture provides a perspective within which ongoing work on RBAC can be synthesized into a common framework.

In conclusion, we note that the appeal of RBAC is in the simplification of the management of authorizations. For example, maintaining cognizance of the permission set of an individual and the consequence of assigning particular role sets to a user is vital. It is also important for a security administrator to know exactly what authorization is implied by a role. This is particularly so when roles can be composed of other roles. Moreover, as new roles and transactions are introduced the security administrator needs tools to assist in their integration into the existing system. Future work in RBAC should identify useful tools for security administration and point the way toward designing these. We feel the central role of the community view in our reference architecture will greatly assist in this objective.

References

- [Bal90] Robert W. Baldwin. Naming and grouping privileges to simplify security management in large database. In *Proceedings IEEE Computer Society Symposium on Research in Security and Privacy*, pages 61–70, Oakland, CA, April 1990.
- [BL75] D.E. Bell and L.J. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, The Mitre Corporation, Bedford, MA, March 1975.
- [CW87] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings IEEE Computer Society Symposium on Security and Privacy*, pages 184–194, Oakland, CA, May 1987.

- [Dep85] Department of Defense National Computer Security Center. *Department of Defense Trusted Computer Systems Evaluation Criteria*, December 1985. DoD 5200.28-STD.
- [FGL93] David F. Ferraiolo, Dennis M. Gilbert, and Nickilyn Lynch. An examination of federal and commercial access control policy needs. In *NIST-NCSC National Computer Security Conference*, pages 107-116, Baltimore, MD, September 20-23 1993.
- [FK92] David Ferraiolo and Richard Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554-563, Baltimore, MD, October 13-16 1992.
- [HRU76] M.H. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461-471, 1976.
- [Lun88] Teresa Lunt. Access control policies: Some unanswered question. In *IEEE Computer Security Foundations Workshop II*, pages 227-245, Franconia, NH, June 1988.
- [LW88] Frederick H. Lochovsky and Carson C. Woo. Role-based security in data base management systems. In C.E Landwehr, editor, *Database Security: Status and Prospects*, pages 209-222. North-Holland, 1988.
- [Mur93] William H. Murray. Introduction to access controls. In Hal F. Tipton and Zella A. Ruthberg, editors, *Handbook of Information Security Management*, pages 515-523. Auerbach Publishers, 1993.
- [Nat92] National Institute of Standards and Technology, and National Security Agency. *Federal Criteria for Information Technology Security, Volumes I and II*, December 1992. Version 1.0, Draft.
- [Nat93a] National Institute of Standards and Technology. *General Security Requirements for Cryptographic Module*, May 24 1993. Draft.
- [Nat93b] National Institute of Standards and Technology, and National Security Agency. *Federal Criteria for Information Technology Security Workshop Proceedings*, July 1993. Issue 1.0.
- [Ora92] Oracle Corporation. *ORACLE7 Server SQL Language Reference Manual*, December 1992. 778-70-1292.
- [PB93] W. Timothy Polk and Lawrence E. Bassham. Security issues in the database language sql. Technical report, National Institute of Standards and Technology, July 30 1993.
- [San88a] R.S. Sandhu. The NTree: A two dimension partial order for protection groups. *ACM Transactions on Computer Systems*, 6(2):197-222, May 1988.
- [San88b] R.S. Sandhu. Transaction control expressions for separation of duties. In *Fourth Annual Computer Security Application Conference*, pages 282-286, Orlando, FL, December 1988.
- [San91] R.S. Sandhu. Separation of duties in computerized information systems. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 179-189. North-Holland, 1991.
- [San92] R.S. Sandhu. The typed access matrix model. In *Proceedings IEEE Computer Society Symposium on Research in Security and Privacy*, pages 122-136, Oakland, CA, May 1992.
- [Ste92] Daniel F. Sterne. A TCB subset for integrity and role-based access control. In *15th NIST-NCSC National Computer Security Conference*, pages 680-696, Baltimore, MD, October 13-16 1992.

- [Te78] D.C. Tsichritzis and A. Klug (editors). The ANSI/X3/SPARC DBMS framework: Report of the study group on data base management system. *Information Systems*, 3, 1978.
- [Tho91] D.J. Thomsen. Role-based application design and enforcement. In S. Jajodia and C.E. Landwehr, editors, *Database Security IV: Status and Prospects*, pages 151-168. North-Holland, 1991.
- [Tin88] T.C. Ting. A user-role based data security approach. In C.E Landwehr, editor, *Database Security: Status and Prospects*, pages 187-208. North-Holland, 1988.

USING THETA TO IMPLEMENT ACCESS CONTROLS FOR SEPARATION OF DUTIES

Rita Pascale
Joseph R. Mc Enerney

Odyssey Research Associates
301 Dates Drive
Ithaca, NY 14850

Abstract

The Trusted Computer System Evaluation Criteria (TCSEC) [5] states that access to an object is based on the access rights of the subject performing the action. In some implementations, access rights seldom change; users are granted rights to an object, and they keep those rights for the lifetime of the object. However, many real-world situations require separation of duties based on dynamic access control, where permission to access an object is dependent on the *state* of the object. This type of state-dependent access control goes beyond TCSEC specifications. One method for specifying dynamic separation of duties is transaction control expressions (TCEs), which are further detailed in [8]. The augmented typed access matrix model (augmented TAM) [1] is one possible way to mathematically represent TCEs. The augmented TAM model is easily implemented in the Trusted HETerogeneous Architecture (THETA), a distributed trusted object-oriented operating system. This paper discusses the problem of separation of duties, and how TCEs, augmented TAM, and THETA provide a simple distributed solution.

1 Introduction

The Trusted Computer System Evaluation Criteria (TCSEC) [5] requires users to be able to specify and control sharing of objects. In the higher assurance systems (e.g., class B3), each object must have a list of authorized individuals who may perform particular actions upon the object. These *access control lists* are often defined statically. For example, all members of group *R* have the right to read an object, and all members of group *W* have the right to write an object. In some real-world examples, this notion of "always having a given right" on a particular object is impractical. Many operations require separation of duties. For example, to protect against em-

bezzlement, banks require that the person who prepares a check cannot be the same person who signs a check [3]. In this example, access is dependent on the previous history of the object. Who prepared the check? That individual must not be permitted to sign the check.

Separation of duty scenarios can be represented with transaction control expressions (TCEs), and in turn, TCEs can be mathematically modeled and refined by the augmented typed access matrix model (augmented TAM). In the following sections, we briefly describe TCEs and augmented TAM; for greater detail, reference [8] and [1]. However, before we explain TCEs and augmented TAM, we will introduce the Trusted HETerogeneous Architecture (THETA) system, which we use to implement a particular separation of duty example. The remainder of the paper details the scenario, which is denoted in TCE format, modeled in augmented TAM, and then implemented in THETA.

2 Introduction to THETA

THETA is a multilevel secure distributed operating system (OS) under development at ORA.¹ The design of THETA is based heavily on the design of Cronus, a distributed OS developed at BBN [13]. THETA currently runs on various trusted platforms including the Sun CMW and the HP-UX BLS. ORA developed the security theory behind THETA's security policy to adhere to the TCSEC guidelines for the Division B, Class 3 criteria.² THETA's approach to security engineering has combined traditional, conservative methods with more advanced, experimental practices. The THETA kernel development emphasized the traditional security approach. It implements the basic communication functions of the Cronus

¹Work on THETA has been sponsored by the Air Force's Rome Laboratory since 1985.

²To truly be B3, the THETA needs to be hosted on a B3 operating system.

kernel, but is completely redesigned and reimplemented. The code that is trusted is minimized. The THETA manager development focuses on *trusted extensibility* [4], an idea that THETA may be developed and adapted to new applications by adding new trusted software. The key THETA advance is the structuring of the manager auto-generation process to simplify the arguments that must be made for security assurance of new trusted managers.

The basic design of THETA is object-oriented: all resources are represented as *objects*. These objects are grouped into *types*, and control of and access to objects of each type is implemented by a *manager* for that type. The manager defines and implements the possible *operations* for the type. Operations are invoked on objects by *clients*³ on behalf of *subjects*. Invocations of operations are communicated from clients to managers by a THETA *kernel* running on each host.

When a client invokes an operation, THETA security is enforced by employing access control checks in managers and the kernel. The security policy for THETA consists of the following:

- A discretionary access control (DAC) policy designed to restrict the use of abstract operations according to the identity of the client.
- A mandatory access control (MAC) policy controlling the flow of information according to the security levels of client and object.
- Policy rules imposing additional constraints such as the definition of the security configuration and the guarantee of trusted paths.

For details see [7], [12], and [11].

The MAC and DAC policies are clearly separated. In fact, they operate at different granularities in the object model. The mandatory policy is enforced at the message passing level. The discretionary policy is enforced at the object level; this policy is global, stating constraints for the entire system rather than for individual hosts.

3 THETA DAC

The access control policy that we describe later in this paper is largely enforced by THETA DAC. THETA DAC is based on the idea of requiring a subject to have certain rights in order to access an object. *Every* THETA object

- is a certain type that can have rights defined for it,
- contains an access control list (ACL), and
- is accessed only by operations that run as procedures in server processes called managers.

³A *client* is a THETA application.

The rights defined for an object are dependent upon the *type* of the object. THETA types are specified in *type definition* files that contain the data structures, the operations that can be performed, the parameters of those operations, and the rights required for each operation. When a subject invokes an operation through a client, THETA demands that the invoking subject possesses the rights specified in the type definition file before the operation will be performed. THETA performs this check by consulting the object's ACL.

The THETA ACL of an object consists of a list of subjects or groups of subjects and a list of access rights granted to those subjects and groups. The THETA machinery for this kind of access control is very general and allows subjects to apply protection measures beyond those described here. The THETA access group set (AGS) mechanism permits subjects to limit their active privileges; the AGS mechanism is further described in [10].

Another point to note is that THETA does not necessarily force a notion of ownership for THETA objects. When a subject creates an object, the subject might not have any rights on the resulting object. Thus creators may not be able to perform *any* operation other than *create*. However, the THETA manager of an object has the power to transform any object in any way, since THETA objects are fully accessible to managers. For our separation of duties example, the manager is the only entity that can enter and delete rights from the ACL.

Now that we have explained THETA and its access control mechanisms, we can detail the separation of duty scenario. The following sections detail transaction control expressions and the access control matrix model that we then implement using the THETA mechanisms.

4 Transaction Control Expressions

Transaction control expressions (TCEs) are written as sequential action-entity pairs. For example, from Ammann and Sandhu's paper [1], the following expression describes a possible process required to issue a check which involves creating and approving a voucher.

```
prepare • clerk
3 : approve • supervisor
issue • clerk
```

What the above notation depicts is that a clerk must first prepare a voucher, then three supervisors must approve the voucher, and then a clerk may issue a check. Any individual who can assume the role of a clerk has the authorization to prepare a voucher or to issue a check, and

any individual who can assume the role of supervisor has the right to approve a voucher. Each step in the TCE is sequential. A check cannot be issued before the associated voucher has been approved three times (this requirement is depicted by the "3:" notation), and likewise, a voucher cannot be approved until it has been prepared.

5 Access Control Model

The above example of issuing a bank voucher does not necessarily explicitly state all of the access control required; it merely states that any individual who can assume a particular role can perform certain actions. What if the access control policy requires that each approval be granted by a unique supervisor? To express the access requirements in finer granularity, we must use an access control model. These models provide a way of representing subjects, objects, and rights and the constraints upon their interactions. The access control model that we use in this paper is the augmented typed access matrix (augmented TAM) mechanism.

In augmented TAM, we model access control by defining a set of *rights* denoted by the set R and a set of *types* denoted by T . There is a subset of the types T that are *subjects*, that is, entities that can perform actions. The remaining types are *objects* that may have actions performed on them. We construct the matrix with one row for each subject and object, and one column for each object, and some subset of access rights are in each matrix cell. To illustrate, when a particular subject *Bob* requests to perform an operation on a particular object *voucher1*, we must first check the matrix cell [*Bob*, *voucher1*] to see if *Bob* has the necessary rights to complete the operation. If he does not, the operation is refused.

Augmented TAM has six primitive commands that are used to manipulate the matrix. They are **enter** $\langle right \rangle$, **delete** $\langle right \rangle$, **create subject**, **delete subject**, **create object**, and **delete object**. Complex access control policies are implemented by combining these commands for manipulating the access control matrix.

Augmented TAM commands consistent with the previous TCE example are given below; they are slightly modified from the examples given in [1]. To specify the commands, we use C for *clerk*, S for *supervisor* and V for *voucher* where *clerk* and *supervisor* are sets of subjects and *voucher* is the set of voucher objects. The matrix notation of [*subject*, *object*] is used to specify a particular subject's access rights on the object. The notation [*object*, *object*] is used to determine the *state* of the object (e.g., has the voucher been approved three times so that it can now be issued?).

```
(a) command begin-prepare-voucher
    ( $C : clerk, V : voucher$ )
    create object  $V$ 
    enter prepare into [ $C, V$ ]
end
(a') command complete-prepare-voucher
    ( $C : clerk, V : voucher$ )
    if prepare  $\in$  [ $C, V$ ] then
        delete prepare from [ $C, V$ ]
        enter prepare' into [ $C, V$ ]
        enter prepare' into [ $V, V$ ]
    end
(b1) command begin-approve1-voucher
    ( $S : supervisor, V : voucher$ )
    if prepare'  $\in$  [ $V, V$ ] then
        delete prepare' from [ $V, V$ ]
        enter approve1 into [ $S, V$ ]
    end
(b1') command complete-approve1-voucher
    ( $S : supervisor, V : voucher$ )
    if approve1  $\in$  [ $S, V$ ] then
        delete approve1 from [ $S, V$ ]
        enter approve1' into [ $S, V$ ]
        enter approve1' into [ $V, V$ ]
    end
(b2) command begin-approve2-voucher
    ( $S : supervisor, V : voucher$ )
    if approve1'  $\in$  [ $V, V$ ]  $\wedge$  approve1'  $\notin$  [ $S, V$ ] then
        delete approve1' from [ $V, V$ ]
        enter approve2 into [ $S, V$ ]
    end
(b2') command complete-approve2-voucher
    ( $S : supervisor, V : voucher$ )
    if approve2  $\in$  [ $S, V$ ] then
        delete approve2 from [ $S, V$ ]
        enter approve2' into [ $S, V$ ]
        enter approve2' into [ $V, V$ ]
    end
(b3) command begin-approve3-voucher
    ( $S : supervisor, V : voucher$ )
    if approve2'  $\in$  [ $V, V$ ]  $\wedge$  approve1'  $\notin$  [ $S, V$ ]
         $\wedge$  approve2'  $\notin$  [ $S, V$ ] then
        delete approve2' from [ $V, V$ ]
        enter approve3 into [ $S, V$ ]
    end
(b3') command complete-approve3-voucher
    ( $S : supervisor, V : voucher$ )
    if approve3  $\in$  [ $S, V$ ] then
        delete approve3 from [ $S, V$ ]
        enter approve3' into [ $S, V$ ]
        enter approve3' into [ $V, V$ ]
    end
end
```

```

(c) command begin-issue-check
    (C : clerk, V : voucher)
    if approve3' ∈ [V, V] ∧ prepare' ∉ [C, V] then
        delete approve3' from [V, V]
        enter issue into [C, V]
    end
(c') command complete-issue-check
    (C : clerk, V : voucher)
    if issue ∈ [C, V] then
        delete issue from [C, V]
        enter issue' into [C, V]
        enter issue' into [V, V]
    end
end

```

The above access control matrix commands complete the specifications for issuing a check. The clerk who prepared the voucher cannot be the same clerk who issues the voucher; each approving supervisor must be unique; the voucher must be at the appropriate stage for each operation (meaning, a voucher cannot be issued before it has been prepared and approved three times).

6 Implementation in THETA

Now that the access control policy is clearly stated by the augmented TAM commands, we can implement the voucher example in THETA by creating the type "voucher". Below is an excerpt of the "voucher" type description, including the three operations and the rights required for each one.

```

type Voucher = ...
    abbrev is voucher
    subtype of Object
    generic rights are
        prepare
    rights are
        approve, issue;
    ...

generic operation PrepareVoucher mac test write
    ...
    requires prepare;

operation ApproveVoucher mac test write
    ...
    requires approve;

operation IssueVoucher mac test write
    ...
    requires issue;

end type Voucher;

```

In this file, we are specifying that the *voucher* type is a subtype of the base type *Object*, which means all applicable attributes of the type *Object* are inherited by all *voucher* objects. Some of the inherited attributes include access rights; for example, the access rights *remove* and *modifyACL* are defined for every THETA object. In addition to these *inherited* rights, we extend the access control by declaring three new rights - *prepare*, *approve*, and *issue*. When specifying the operation semantics, we state what rights are required for each operation. Therefore, for a subject to successfully invoke the PrepareVoucher operation, that subject must have the *prepare* right for *voucher* objects as is specified in the file above.

In addition to declaring the DAC rights necessary to perform an operation, we indicate what MAC security test must be passed. The mandatory access test for each operation in this example is a "mac test write", meaning that the security level of the *subject* must be *dominated by* the security level of the *object*. Other possible mandatory access tests are "mac test read" where the security level of the subject must dominate the level of the object, and "mac test readwrite" where the level of the subject and the object must be equal. These MAC tests follow the standard Bell-LaPadula rules, see [2] for details.

For an invocation to complete successfully, it must pass three levels of access checks. First, the THETA kernel performs the MAC security checks to guarantee that the security level of the invocation is within the level range of the Voucher Manager. The Voucher Manager ensures that the invocation level bears the correct relationship to the object.⁴ Next, the Voucher Manager requests any needed DAC information from the Authentication Manager. Using this information, the Voucher Manager makes sure that the invoker has the privilege to perform the operation on the target object. Finally, having correctly passed the MAC and DAC checks, the operation code is executed by the Voucher Manager and any further access checks and restrictions are performed there. In our example, these additional conditions include operation ordering and invoker uniqueness requirements.

To continue our example, we grant users in the *clerk* role the *prepare* and *issue* rights, and users in the *supervisors* role the *approve* right. Normal THETA subjects do not automatically have any access rights to any objects. As the privileged THETA administrator, we create the groups *clerk* and *supervisor* and then grant those groups the rights associated with those roles. To acquire the access rights associated with a group, a member must bind the appropriate *access group set* to an invoking client; the AGS mechanism is further described in [10].

⁴By this, we mean that the Voucher Manager enforces the no write-down and no read-up rules, and possibly more restrictive access depending on the semantics of the operation.

7 Separation of Duty Mechanism

Assuming the subject has invoked the operation at an acceptable security level and the subject possesses the necessary access rights, the next step is for the manager to check the object to see if it is in the appropriate state. For example, the manager should not permit a voucher to be issued before it has been approved three times. The manager checks the state by consulting the voucher object's ACL and the internal status information of the object. When a voucher is created, the manager sets the ACL of the new voucher to require the "approve" right for the next operation; when the voucher has been approved three times, the manager changes the ACL to require the "issue" right; and once issued, the manager removes all access rights. The internal status information is used to count the number of times a voucher has been approved and to check what users have acted on this voucher previously.

The THETA C code for the "issue voucher" operation that corresponds to the TAM commands "begin-issue-check" and "complete-issue-check" is

```

voucherIssueVoucher(r, input, output)
  OperationParms *r;
  reqvoucherIssueVoucher *input;
  repvoucherIssueVoucher *output;
{
  voucher          *Voucher;
  ACLEntry         TheACL;

  /*
   * Prior to getting to this step, THETA DAC
   * mechanisms ensure that the user that
   * invoked this operation has the necessary
   * "issue" right for the specified object.
   */

  /*
   * Retrieve the object data from the THETA
   * database.
   */
  if ((Voucher = (voucher *)
      GetVarData(VOUCHER, r->objdes)) == NULL) {
    Nack(r, E_NOOBJ);
    return;
  }

  /*
   * To issue a voucher, it had to be prepared,
   * and then approved by three distinct
   * managers. Successful completion of these
   * steps is indicated by the "Status" field
   * of the voucher.
   */

```

```

if (Voucher->TAMcheck.status != APPROVED3) {
  FreeVarData((voucher *) Voucher);
  Nack(r, E_BAD_OPER_FOR_OBJ);
  return;
}

/*
 * To ensure separation of duties, we must
 * make sure the clerk who is attempting the
 * issue is not the same clerk who prepared
 * the voucher.
 */
if (IsSameUID(&r->msgdes.principal,
  &Voucher->TAMcheck.preparer) == TRUE) {
  FreeVarData((voucher *) Voucher);
  Nack(r, E_DUTY_SEPARATION_CONFLICT);
  return;
}

/*
 * If the voucher is in the proper state
 * (i.e., APPROVED3), and the user doing the
 * operation is not the same user that did
 * the prepare, issue the voucher.
 */
Voucher->TAMcheck.issuer =
  r->msgdes.principal;
Voucher->TAMcheck.status = ISSUED;

/*
 * Now that the command is complete, remove
 * the "issue" rights from the voucher's
 * access control matrix to ensure that some
 * other clerk does not try to repeat the
 * issue command.
 */
TheACL.principal_uid = ClerkGrpUID;
TheACL.rights = R_issue;
RemoveFromACL(&r->objdes->acl, &TheACL);

/*
 * Save the modified voucher object in the
 * THETA database.
 */
PutVarData(VOUCHER, r->objdes,
  (char *)Voucher);

/*
 * Notify the calling function that the
 * operation completed successfully.
 */
output->valid = TRUE;
}

```

The other TAM commands are implemented in a simi-

lar manner. The Voucher Manager performs the MAC and DAC checks before the operation code is executed, and within the operation code, the manager checks the status field of the object in order to enforce the "unique operator" requirements.⁵ Using the flexible access control mechanisms of THETA, we were able to implement the augmented TAM model for separation of duties scenarios.

8 Model Versus Implementation

In the TAM commands, there is a distinction between the beginning of an action and its completion, and the status of an operation is tracked by consulting the rights in the access control matrix. For the implementation in THETA, the begin and complete steps collapse into a single operation; therefore, we do not make a distinction, for example, between *prepare* and *prepare'*.

The access control matrix contains a cell for each subject-object pair so that when a subject attempts an operation on a specific object *V*, the subject's access rights can be determined easily. This portion of the matrix is a direct correlation to the THETA ACL mechanism for objects. The access control matrix also contains a cell for each object-object pair in order to track the current state of the object. For example, in the augmented TAM commands, the "enter *issue'* into [*V*, *V*]" step indicates that the object *V* has been issued and should therefore not be permitted to be issued again. In the implementation, this historical data is maintained within the object itself and the Voucher Manager permits accesses based on that data.⁶ The THETA DAC mechanism also assisted in tracking the status of the object. For example, once a voucher object is issued, the *issue* right is removed from the object's ACL, and therefore the voucher cannot be issued again. The historical data maintained in the object is mainly used to enforce the "unique operator" requirement between operations. That is, when a supervisor attempts to approve a voucher, the THETA Voucher Manager consults the historical data of the voucher object to ensure that this supervisor has not previously approved this same voucher.

⁵By this, we mean the operation code within the manager checks the identities of all subjects who have performed operations on this particular object previously and stops a subject from performing two operations on a single voucher object.

⁶In this particular implementation, we ensure that operations are invoked in the proper sequential order; the amount of time elapsed between each step is not taken into account. By adding a timestamp to the historical data, we could easily add time limits to each step and to the overall "issue check" process.

9 Multilevel Secure Case

The notions expressed by TCEs and implemented in THETA can be extended to the case of multilevel operation by adding an operating level. As an example, we consider the case of preparing, approving and upgrading a document from one security level to a higher security level. In order to satisfy MAC policy, information can flow up by blindly writing information up to a higher level repository whose very existence is uncertain, or by reading a low-level repository from a high level.

The first method involves uncertainty about the success of the write up which can be offset by having a cleared user check the results at the high level at a later time. The second approach leaves open the possibility that the low data may change during the read down; this is called a "dirty read". A solution to the "dirty read" is to lock the object at the low-level, read the data, and then release the low-level lock; however, this solution introduces an unacceptable information flow. For this example, we have chosen the first approach.

Since the multilevel case involves a notion of security level, we encapsulate ordinary TCEs in square brackets and assert that they occur at a level. The set of information system security officers is denoted by *isso* while *clerk* and *supervisor* are as before. We summarize the document upgrade as follows:

```
[prepare • clerk] @level
[approve • supervisor] @level
[upgrade • level • isso] @level
```

The third TCE step contains the final higher level that the document will reach. We stipulate that the sets *isso*, *supervisor* and *clerk* are pairwise disjoint. A particular instantiation of these TCEs could be

```
[prepare • Phydeaux] @SECRET
[approve • Ropher] @SECRET
[upgrade • TOPSECRET • Phang] @SECRET
```

where *Phydeaux* is a *clerk*, *Ropher* is a *supervisor*, and *Phang* is an *isso*.

Before presenting the TAM commands corresponding to these TCEs, we declare a set of TAM command extensions that further elaborate the specification and bound any implementation.

- An implicit security level, at which a TAM command is performed, is denoted by *cmdlev*.
- An access matrix at each security level *L*, and each matrix element is denoted [*X*,*Y*,*L*].
- We permit write up operations. These assume that no information flows back down to indicate success

or failure of such actions. This is an information upgrade mechanism that is enabled by an *upgrade* right.

- No subject can run at more than a single security level at a time. Therefore, a command cannot change its level of operation.
- Members of *isso* are cleared for all levels but can execute a command only at a single level.

- (a) command begin-prepare-document
 ($C : clerk, D : document, L : level$)
 if $cmdlev \leq L$ then
 create subject $D @ L$
 enter *prepare* into $[C, D, L]$
 end
- (a') command complete-prepare-document
 ($C : clerk, D : document, L : level$)
 if $cmdlev = L \wedge prepare \in [C, D, L]$ then
 delete *prepare* from $[C, D, L]$
 enter *prepare'* into $[C, D, L]$
 enter *prepare'* into $[D, D, L]$
 end
- (b) command begin-approve-document
 ($S : supervisor, D : document, L : level$)
 if $cmdlev = L \wedge prepare' \in [D, D, L]$ then
 delete *prepare'* from $[D, D, L]$
 enter *approve* into $[S, D, L]$
 end
- (b') command complete-approve-document
 ($S : supervisor, D : document, L : level$)
 if $cmdlev = L \wedge approve \in [S, D, L]$ then
 delete *approve* from $[S, D, L]$
 enter *approve'* into $[S, D, L]$
 enter *approve'* into $[D, D, L]$
 end
- (c) command begin-upgrade-document
 ($SO : isso, D : document,$
 $L_{low} : level, L_{high} : level$)
 if $cmdlev = L_{low} \leq L_{high}$
 $\wedge approve' \in [D, D, L_{low}]$ then
 create subject $D^{up} @ L_{high}$
 copy D up into D^{up}
 delete *approve'* from $[D, D, L_{low}]$
 enter *upgrade* into $[D^{up}, D^{up}, L_{high}]$
 enter *upgrade* into $[SO, D^{up}, L_{high}]$
 end
- (c') command complete-upgrade-document
 ($SO : isso, D : document, L_{high} : level$)
 if $cmdlev = L_{high} \wedge upgrade \in [D, D, L_{high}]$
 $\wedge upgrade \in [SO, D, L_{high}]$ then
 delete *upgrade* from $[D, D, L_{high}]$
 delete *upgrade* from $[SO, D, L_{high}]$

enter *upgrade'* into $[SO, D, L_{high}]$
 end

As shown in the single level case, the THETA system can also support multilevel TCE/TAM specification without difficulty. The THETA automatic code generation facilities handle a large portion of the multilevel aspects of the above example; thus, most of the burden of developing multilevel applications is removed.

10 Comparisons with Other Systems

It is possible to implement separation of duty scenarios through type enforcement mechanisms as found in the LOCK⁷ system [6] and the ECS subsystem⁸ [9]; however, the implementation is much more complicated.

Type enforcement controls data accesses based on a "domain-type table." There is a type associated with each object and a domain associated with each subject, and a table (much like the subject-object matrix used in our example) which states the rights that subjects of particular domains have on objects of particular types.

LOCK's limitation in the implementation of the separation of duties scenario is that it does not handle dynamic changes in the domain-type table. A single voucher object cannot be prepared, approved and then issued. At each step, the object is transformed into a different object type in order to grant access rights to subjects of different domains and deny subjects of other domains. Also, another complication in our scenario is the "unique operator" requirement; it is not obvious how one would implement the requirement of "each operation must be performed by a distinct user" with LOCK's type enforcement mechanism.

The ECS system handles dynamic access controls by associating a finite state automata on the object. The finite state automata contains information about the current state, operations that can be performed to put the object into a different state, and a "separation of duty" specifier (which holds information about what subjects can and cannot perform operations on the object). With some administrative overhead, the ECS system handles the separation of duties scenario; however, THETA has the advantage of being a distributed, heterogeneous solution.

⁷Work on the Logical Coprocessing Kernel (LOCK) is being done by Secure Computing.

⁸The Extended Access Control Subsystem (ECS) is an untrusted TCB subset for use with an extended version of the Trusted Xenix operating system. Work on ECS was done by Trusted Information Systems. Xenix is a trademark of the Microsoft Corporation.

11 Conclusions

The combination of transaction control expressions and augmented typed access matrix commands provide a rich specification language for defining the semantics of applications that require separation of duties and operation ordering. In both the single level and multilevel cases, THETA offers a convenient distributed computing environment that facilitates implementation of solutions to this kind of problem.

THETA handles the separation of duties problem easily because of very adaptable access control mechanisms that go beyond the TCSEC specifications. The object-oriented nature of the system provides developers the opportunity to make operations as restrictive or as open as necessary. In the type specification files, developers can define new rights for a type and state which rights are required for each operation. The specification files are then used to autogenerate the code to enforce the rights requirements. The developer can then use the operation code to further restrict access. We demonstrated this extension of access control in our example by enforcing unique subjects to perform each step of the "issue check" operation.

The TAM checks required by the TCEs are enforced in the THETA implementation by

- rights of invoking subjects,
- THETA ACLs on objects,
- internal object status information, and
- rights to modify ACLs are restricted to the Voucher Manager.

By verifying a subject's rights on objects, the Voucher Manager ensures that only subjects of the appropriate groups perform particular operations; that is, only clerks may prepare and issue a voucher, and only supervisors may approve a voucher. The manager consults the THETA ACLs on objects to force the operations to be in the proper sequential order of prepare, approve, and then issue. The internal historical data of an object is used to restrict each step of the "issue check" operation to unique subjects. Restricting the right to modify ACLs to the Voucher Manager protects the integrity of the "issue check" process. This restriction is necessary because the ACL of a voucher object helps track the state of the voucher. This means, when a voucher object has the "issue" right in the ACL, the "issue" operation can be performed; thus, if a clerk could randomly add the "issue" right to a voucher's ACL then the approval process is effectively subverted.

Implementing the augmented TAM model in THETA was a simple exercise that took less than a day. THETA's

flexible access control mechanism easily handles the separation of duty problem. The cooperation of the THETA kernel, manager, and access control mechanisms provide a flexible and secure solution to the separation of duty scenario. In addition, THETA provides a trusted *distributed* solution to such problems that may be very useful in secure environments.

References

- [1] Ammann, P. E., Sandhu, R. S., "Implementing Transaction Control Expressions by Checking for Absence of Access Rights," *Proceedings of the Eighth Annual Computer Security Applications Conference*, December 1992.
- [2] Bell, D., LaPadula, L., "Secure Computer Systems: Unified Exposition and Multics Interpretation," MTR-2997, Mitre, Bedford, Massachusetts, 1975.
- [3] Clark, D., Wilson, D., "A Comparison of Commercial and Military Computer Security Policies," *IEEE Symposium on Security and Privacy*, 1987.
- [4] McEnerney, J., Weber, D., Brown, R., and Varadarajan, R., "Automated extensibility in THETA", *Proceedings of the 13th National Computer Security Conference*, October 1990.
- [5] National Computer Security Center, Fort Meade, MD, *Trusted Computer Systems Evaluation Criteria*, DoD 5200.28-STD, December 1985.
- [6] O'Brien, R., and Rogers, C., "Developing Applications on LOCK," *Proceedings of the 14th National Computer Security Conference*, October 1992.
- [7] Proctor, N., Wong, R., "The security policy of the Secure Distributed Operating System Prototype," *Proceedings of Fifth Aerospace Computer Security Applications Conference*, December 1989.
- [8] Sandhu, Ravi S., "Transaction Control Expressions For Separation of Duties," *Proceedings of the Fourth Annual Computer Security Applications Conference*, December 1988.
- [9] Sterne, Daniel S., "A TCB Subset for Integrity and Role-Based Access Control," *Proceedings of the 15th National Computer Security Conference*, October 1992.
- [10] ORA, "Software Programmer's Manual for the Experimental Secure Distributed Operating System Development," Technical Report, THETA CDRL No. A011, July 1991.

- [11] ORA, "Software Requirements Specification for the Experimental Secure Distributed Operating System Development," Technical Report, THETA CDRL No. A008, July 1991.
- [12] ORA, "Formal Model for the Experimental Secure Distributed Operating System Development," Technical Report, THETA CDRL No. A009, July 1991.
- [13] Schantz, R., Thomas, R., and Bono, G., "The architecture of the Cronus distributed operating system," *Proceedings of the 6th International Conference on Distributed Computing Systems*, May 1986.

IMPLEMENTING ROLE BASED, CLARK-WILSON ENFORCEMENT RULES IN A B1 ON-LINE TRANSACTION PROCESSING SYSTEM¹

Barbara Smith-Thomas
AT&T Bell Laboratories, W3-H63, AT&T Guilford Center, P.O.Box 26073,
Greensboro, N.C. 27420

Wang Chao-Yeuh
Wu Yung-Sheng
Institute for Information Industry, 13th Floor, 216 Tun Hwa South Rd., Sec 1,
Taipei, Taiwan, ROC

ABSTRACT

This paper describes an implementation of role-based discretionary access controls in an on-line transaction processing system that supports commercial banking applications. These discretionary controls together with the underlying B1 secure operating system implement the "E" rules of the Clark-Wilson Integrity model. The system described is currently in beta test.

Keywords: access control, discretionary access control, mandatory access control, role, security, on-line transaction processing, Clark-Wilson integrity, access control list, capability

1. Introduction

In the past 10 years it has become clear that the TCSEC [13] does not adequately address computer security requirements for integrity. This is because the initial US customer for secure computer systems was the Department of Defense, and the paper-world standards for protection against disclosure of National Security classified information were well understood and relatively amenable to translation into the currently accepted label-based Mandatory Access Control requirements. However, the requirements for integrity protection were less well understood at the time. Even now, standards for integrity are less well defined.

One influential effort to model requirements for integrity was published by Clark and Wilson in 1987 [3]. Clark and Wilson separate their requirements into two sets: the C, for "certification", and the E, for "enforcement" rules. The Clark-Wilson Integrity model provides for two types of data, the CDI and UDI, or "constrained data item" and "unconstrained data item", respectively. There are two types of operations, the IVPs, or Integrity Validation Procedures, and the TPs, or Transaction Procedures. The CDI's are the high integrity data that have been validated by one or more IVPs and that can only be modified by TPs. The "C" rules capture the correctness properties that the applications must be certified to have; the "E" rules capture the access controls

1. The work described in this paper was sponsored by the Ministry of Economic Affairs of the Republic of China.

that must be enforced by the system. See [7] and [14] for additional information on the Clark-Wilson model.

The Institute for Information Industry of the Republic of China has undertaken a project to produce a B1 certifiable Automated Banking System by enhancing an existing commercially available system. As the risk analysis and design phases of the project progressed, we realized that the MAC security controls we were designing did not adequately capture the integrity requirements of a banking environment. We decided to adopt a mixed strategy in which B1 (Mandatory) label-based access controls would be enforced by the underlying operating system and database system, and (Discretionary) identity-based controls would be applied within the Automated Banking System to provide a finer granularity of control for individual transactions and database accesses. The resulting system is B1 certifiable and implements the Clark-Wilson enforcement rules.

2. The UCP² On-Line Transaction Processing System

Figure 1 shows the basic layered architecture of the Automated Banking System. It consists of the "User Control Program" (UCP) on-line transaction processing platform along with a collection of application programs, the transactions, that provide the banking specific actions such as making a deposit to a passbook savings account.

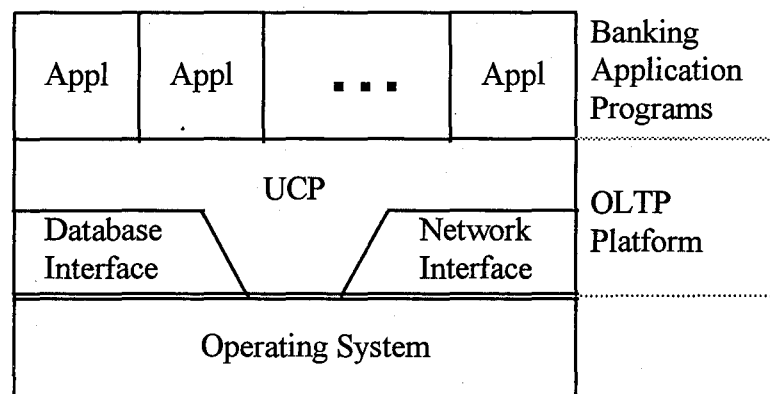


Figure1: Simplified ABS Architecture

UCP provides both a convenient interface for programmers and users of a banking or other database system, and run-time management of the associated database interactions. The original UCP runs on UNIX³ SVR4.0 without extra privilege. It includes a menu interface, transaction dispatchers, and a network interface. Database accesses are made by the transaction dispatchers on behalf of the transactions, although there is no mechanism by which direct accesses to the database by an application can be prevented. UCP has the ability to handle both local transactions and transactions which actually execute at a remote site.

2. UCP is a trade mark of the Institute for Information Industry

3. UNIX is a registered trade mark licensed exclusively by X/Open, Ltd.

The underlying operating system for the B1 certifiable version of the Automated Banking System is System V/MLS⁴. The labeling enhancements to UCP are described in the companion paper "B1 Security in an On-Line Transaction Processing System -- A Project's Experience" [12]. In this paper we describe the discretionary and role based enhancements to UCP that, in conjunction with the label based enhancements, implement the E rules of the Clark-Wilson integrity model.

3. Basic Strategy

When designing access control mechanisms, the first issues that must be addressed are the definitions of subjects and objects, and the types of access provided by the system. In UCP we define two types of subjects: the users of the system and the transactions that execute on behalf of those users. Untrusted users can only execute transactions; they have no other access rights within UCP. The transactions, in turn, access other objects on behalf of the user. The objects that a transaction can access are the application specific data contained in the database files maintained by UCP, and other transactions that can be directly invoked by a given transaction. In either case the access is under the control of UCP. The access modes at the database file level are Read, Update, Insert, Delete and at the field granularity are Read and Update only. The only access mode for a transaction is Execute.

The most general form of representation of access rights information is an access control matrix. Access Control Lists (ACLs) and Capability Lists (CLs) can both be viewed as compact methods for representing the sparse Access Control Matrix. ACLs represent the access control information by object, that is by column of the matrix, and are generally part of the definition of the object. CLs represent the access control information by subject, that is by row, and are generally associated with the executing process. In the NCSC guidelines for implementing discretionary access control in trusted systems [15], ACLs are recommended as the most flexible and usable way to implement DAC. So, the original blueprint for UCP DAC was to put ACLs on all UCP objects, both transactions and application data (figure 2(a)). But after some discussion, another approach, the ACL and CL combination associated with the transaction, which is shown in Fig 2(b), was proposed and adopted. The reasons for this decision are:

- UCP is a transaction based platform, so putting the access control information on the transactions seems to be a natural approach.

- Since a search for access control information is a necessary step for each access mediation, simplifying the search algorithm is an important consideration. Storing the database access control information with the transaction in a CL makes the information immediately available when the access request is made. It is not necessary to consult the database for the access information.

- In the NCSC guidelines for implementing DAC, Capability Lists are criticized for design problems with revocation of access from individual users. Since in a general computer system the passing of capabilities from one user to another is not controlled and capabilities can be stored, determining all the subjects who have access to a particular object generally is not possible. This

4. System V/MLS is a registered trade mark of AT&T

makes the revocation of access difficult, especially for archived data. It is also difficult with capability lists to implement access for named groups. In UCP, these problems do not occur. The behavior of the transaction with respect to data access is determined at design time; we do not allow them to be changed at run-time. In addition, capabilities cannot be passed from one transaction to another. So which transactions have access rights to a database object is static and easily determined.

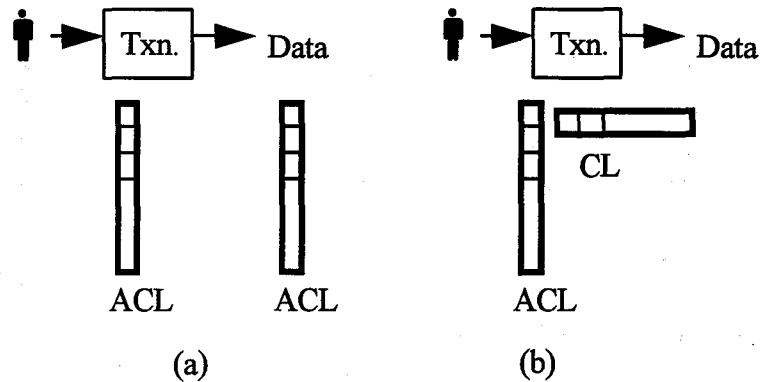


Figure 2. Alternate Mechanisms for UCP Access Controls

4. User Attributes (Roles)

The major disadvantages of ACL's are potentially large storage space requirements, and difficulties with revocation of access, especially for archived data. Both of these problems can be addressed by some sort of grouping mechanism. In a commercial context the natural grouping is a "role". The conceptual ideas of a role are:

- Roles shall be associated with a business. UCP applications are designed by business. A business consists of several related transactions sharing the same data format, the same data files, etc. Usually, a transaction is designed to be executed by the people who occupy a particular position in that business, e.g. the tellers of the PassBook business. UCP roles are an attempt to represent this existing grouping.

- There are several positions under the same business. Combining the business and the position, we can identify a role uniquely. In the following description, we refer to it as a UCP role or a position within a business. The positions in one business are independent of the positions in other businesses. Initial values are chosen arbitrarily except that the role of business administrator is recognized by UCP. A user with this position has the authority to set and modify the ACLs of the transactions in the same business as the administrator. A user may be a member of several UCP roles, both within the same business and in different businesses. The UCP administrator assigns roles to users. When separation of duties is desired, it is the responsibility of the UCP administrator to set up the users' roles appropriately. If a user belongs to more than one position in a business, the user can switch between those positions by explicit action.

A file containing (user, role) pairs will be defined. We will say a user is "authorized" for a role if there is a (user, role) pair in this file.

5. Transaction Execution

In UCP, a transaction can be invoked in two ways: (1) by a user from a menu via the menu interface or (2) by another transaction. Each transaction known to UCP is assigned a unique four character transaction identifier, also called a TxnID. The first character of the TxnID identifies the business; the remaining characters, which are usually digits, identify the particular transaction within the business. A user is identified to UCP by a triple (userID, Current Role, nodeID), an internal user identifier, the current role assigned by UCP, and an identifier for the network node at which the user is working.

5.1. Transaction Invocation by a User

Each transaction has three ACLs associated with it: the User ACL, the 2nd Authorization ACL, and the Transaction ACL. The User ACL specifies which users can invoke the transaction via the menu interface. The User ACL consists of a list of security identifiers (UserID, Role, NodeID), as described above. Each of these security identifiers is also called an "Access Control Element", or ACE. In order to reduce the length of the ACL we define some wildcards that can be used in an ACE:

- "*" : used in UserID, Role, or NodeID means all tellers, positions, or nodes.
- "#" : used in NodeID means the local node.
- "\$" : used in NodeID means all nodes except the local node.

We restrict execution of a transaction to users who are authorized for the business of the transaction. In UCP, some transactions execute at a different node from the one at which it is invoked by the user. Those transactions will have an ACL at both the user and the transaction side of the connection. When a user executes a remote transaction, he or she must pass both sites' mediation: once at the menu interface of the local node and again at the server of the remote node.

An example may be useful. The following figures illustrate the sequence of access checks involved in a user's executing a transaction. Assume that there are two businesses, B1 and B2. Assume further that the user User01 is authorized for role 1-050, B1 clerk, and has logged in at Node 001.

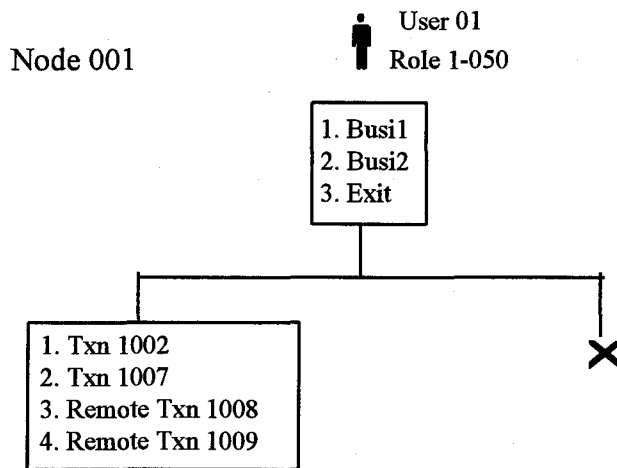


Figure 3: Selecting a business

Once the process of signing on to UCP is complete, the user is presented with a top level menu from which the desired business is selected. From the information in the user information file, the menu interface determines that the user is authorized for role 1-050. So selection of choices 1 or 3 will succeed. If the user selects business 2 a "permission denied" message will be displayed. Assume that the user selects business 1. The menu interface will set the user's current security attributes to User01.1-050.# and present the initial menu from business 1. Suppose, now, that the user ACLs on the four business 1 transactions are shown in figure 4.

<u>Txn 1002</u>	<u>Txn 1007</u>	<u>RmTxn 1008</u>	<u>RmTxn 1009</u>
01. * #	01.1-100.#	* .1-050.#	* .1-050.#
04. * #	* .1-200.#		
* .1-050.#			
* .1-150.#			

Figure 4: Business 1 User ACLs at Node 001

The user will be permitted to select transactions 1002, 1008, and 1009 because the user's security identifier matches one or more ACE entry in the user ACL of the transaction. The user will not be permitted to select transaction 1007, even though the user's UserID appears in an ACE for that transaction, because the user is not operating with the appropriate Role for that transaction. Note that UCP discretionary access control is done when the user selects a transaction. For a remote transaction, the server will mediate discretionary access a second time using the ACLs on the remote host. Both mediations must pass for access to be granted. For example, suppose that in addition to the ACLs on Node 001 given above, the following ACLs are defined at Node 002.

<u>Txn 1008</u>	<u>Txn 1009</u>
.1-050.	*.1-0500.003
	*. * .005

Figure 5: Business 1 User ACLs at Node 002

The user will be allowed to select either transaction 1008 or 1009 at node 001. The user's identity will be passed to the remote host along with the transaction request. Execution of transaction 1008 will be permitted, but execution of transaction 1009 will be denied since the user is at node 001, not node 003 or 005.

5.2. 2-Person Transaction Control

In the current banking applications supported by UCP, some transactions require 2-person control. For example, when a teller executes a "withdraw" transaction, if the amount to be withdrawn is greater than some fixed limit a supervisor must enter his or her password into the transaction input panel to approve the transaction. To support 2-person control, a second user ACL is provided. This ACL has the same format as the primary ACL, but it is referenced only by the menu interface check routine that is responsible for the second person's identification and authentication.

5.3. Transaction Invocation by Another Transaction

In UCP a transaction can be invoked by another transaction. Because we also wish to have UCP controls on execution of a transaction by another transaction, the ACL associated with a transaction will include a set of transaction entries in addition to the two sets of user entries. A transaction ACE will contain the TxnID of the invoking transaction, the node(s) from which the execution request can be accepted, along with other information meaningful to UCP called the "execution mode". A transaction can be invoked by a transaction in another business. This provides us with the flexibility to have transaction processing cross business boundaries, but only through a tightly controlled mechanism.

6. Application Data Access

Up to now we have concentrated on determining which users can execute what transactions. The Clark-Wilson E-2 rule requires that the data accessed by a transaction must be specified as well. In order to access the contents of a database file, a transaction must be explicitly authorized to do so by having the file listed in the capability list associated with the transaction. Within some business files, particularly sensitive fields may require additional protection. For example, it may be the case that only selected Passbook transactions should be able to update the account balance. This field needs to be further constrained. Constraint files are defined to store this additional constraint information. Each entry in one of these files will contain the name of the file containing the constrained fields, the names of the constrained fields, and other information used by the system to locate the fields within the database files. In the PassBook example above the constraint file might contain the following entry declaring that the PBM_BAL field in the PB_ACCT file is constrained.

```
PB_ACCT    PBM_BAL    <location information>
```

Any fields not listed as constrained are accessible to all transactions that are authorized to access the file.

A Capability List (CL) consists of one or more Capability Elements (CEs); each Capability Element consists of a filename and a set of file access modes, followed by 0 or more filename/field access mode pairs. Each business file accessed by a transaction must be listed in the transaction's CL. If the file contains any constrained fields that must be accessed by the transaction, those fields must also be listed in the CE that lists the file.

The Clark-Wilson E-2 rule: The system must maintain a list of which users can access what data via which transactions, explicitly provides for multiple capability lists depending on the identity of the invoking user. However, there is no loss of generality in requiring all authorized users of a transaction to access the database in the same way as long as executable code can be shared between transactions. If two sets of users of some piece of code must access data differently, two txnID's and hence two ACLs and two Capability lists can be defined for that code.

7. Propagation Modes

When designing an Access Control mechanism, defining who has the authority to grant and revoke access permissions is as important as defining the access rules. In the NCSC discretionary access control guidelines [15], four access propagation modes are presented: Hierarchical, Owner, Laissez-Faire, and Centralized. The "Owner" option is the one most commonly intended when discretionary access control is discussed; in this option the owner of an object has the authority/discretion to set the access modes on the object. Discretionary access controls also carry the assumption that the controls protect the container rather than the information. More recently, access controls have been classified as mandatory, discretionary, and non-discretionary [1], implying some kind of centralized administration. It is perhaps more useful to abandon the notions of "Discretionary" and "Mandatory" entirely and instead classify access controls by attributes such as label based or identity based, centrally administered or owner administered, protecting the container or the data, etc.

In UCP we adopt a restricted hierarchical control for the ACLs defining access by users to transactions and centralized control for the capability lists defining the accesses by transactions to data. Originally we adopted a completely centralized mode of control for both ACL's and CL's. We expected that the access rights of a transaction would be determined when the transaction is installed in UCP and would not change thereafter. That is, who can execute the transaction and what data the transaction can access is only determined by the system administrator who is the "owner" of the all UPC transactions and data. However, in a commercial environment more flexibility is needed in authorizing users to execute transactions. For example, a supervisor may go on vacation and a senior teller may temporarily need to perform that supervisor's duties. In order to support additional flexibility, we add a limited hierarchical control on the transaction ACLs. Selected users who are authorized by the system administrator have the right to grant or revoke transaction execution rights to other users. The ability to grant access cannot be further delegated.

8. Other Clark-Wilson E-rules

So far we have concentrated on the Clark-Wilson E-2 rule. There are three other E rules.

E-1) CDIs can only be changed by authorized TPs.

The MAC controls of System V/MLS are used to limit access to the databases to transactions registered with UCP. Two new categories, UCP and UCP_Private, are defined when Secure UCP is installed on an MLS system. All files associated with UCP: transactions, databases, utilities, and UCP itself, are marked with the UCP category; the databases under the control of UCP are also marked with the UCP_Private category. Authorized UCP users are cleared to the UCP category, but not to the UCP_Private category. Transactions execute with the identity and, usually, with the login label of the invoking user. Database accesses are done by UCP itself, which "promotes" the user's label with the addition of the UCP_Private category before initiating a database access with the identity of the user. Thus database accesses are limited to users cleared to UCP, using transactions known to UCP, and with the individual database actions mediated by UCP.

E-3) Users must be authenticated by the system

This rule is enforced by the underlying System V/MLS operating system.

E-4) Only authorized system officers may change the access information.

Only the UCP administrator is authorized to add transactions to the system or to change the Capability List or transaction ACL associated with a transaction. The UCP administrator and the business administrators are authorized to change the user ACL and second authorization ACL associated with a transaction.

It should be noted that Secure UCP respects the MAC and DAC mechanisms provided by System V/MLS and the databases, and provides an audit mechanism that complements the audit trail of System V/MLS.

9. Conclusion

Using a combination of label mechanisms, and centrally administered ACL and capability list mechanisms, the security enhanced version of UCP provides a MECHANISM for implementing Clark-Wilson type CDI - TP applications. According to Secure UCP itself, the transactions and databases are "untrusted" -- that is, operate without privilege. The UCP administrator is responsible for certifying the transactions that he/she installs. The UCP administrator and the various business administrators are also responsible for maintaining the access control tuples on which UCP bases its access control decisions.

10. References

[1] Abrams, M., "Renewed Understanding of Access Control Policies," Proceedings of the 16th National Computer Security Conference, 1993.

- [2] Abrams, M., Eggers, K., Lapadula, L. and Olson, I., "A Generalized Framework for Access Control : An Informal Description," Proceedings of the 13th National Computer Security Conference, 1990, 135-143.
- [3] Clark, D. and Wilson, D., "A Comparison of Commercial and Military Computer Security Policies," Proceedings of the Symposium on Security and Privacy, IEEE, 1987, 554-563.
- [4] Downs, D., Rub, J., Kung, K. and Jordan, S., "Issues in Discretionary Access Control," Proceedings of the Symposium on Security and Privacy, IEEE, 1985, 208-218.
- [5] Ferraiolo, D. and Kuhn, R., "Role-Based Access Controls," 15th National Computer Security Conference, 1992, 554-563.
- [6] Graubert, R., "On the Need for a Third Form of Access Control," Proceedings of the 12th National Computer Security Conference, 1989, 296-303.
- [7] Jueneman, R.R., "Integrity Controls for Military and Commercial Applications," Proceedings of the Aerospace Security Conference, IEEE, 1988.
- [8] Karger, P.A., "Implementing Commercial Data Integrity with Secure Capabilities," Proceedings of the Symposium on Security and Privacy, IEEE, 1988.
- [9] Lee, T.M.P., "Using Mandatory Integrity to Enforce 'Commercial' Security," Proceedings of the Symposium on Security and Privacy, IEEE, 1988.
- [10] Lipner, S. B., "Non-Discretionary Controls for Commercial Applications," Proceedings of the Symposium on Security and Privacy, IEEE, 1987.
- [11] Shockley, W.R., "Implementing the Clark/Wilson Integrity Policy Using Current Technology," Proceedings of the 11th National Computer Security Conference, 1988, 29-37.
- [12] Smith-Thomas, B., Liu, S., and Wu, J., "B1 Security in an On-Line Transaction Processing System -- A Project's Experience," submitted for publication, 1994.
- [13] *Department of Defense Trusted Computer System Evaluation Criteria*, U.S. Department of Defense Standard, DOD 5200.28-STD, December 1985.
- [14] *Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS)*, Bentley College, 1987.
- [15] *A Guide to Understanding Discretionary Access Control in Trusted Systems*, NCSC, September 1987.

VIRTUAL VIEW MODEL TO DESIGN A SECURE OBJECT-ORIENTED DATABASE

N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, K. Yazdanian

ONERA / CERT
2 avenue Edouard Belin
31055 Toulouse cedex
France

Email : nora@tls-cs.cert.fr, cuppens@tls-cs.cert.fr, gabillon@tls-cs.cert.fr, yazdanian@tls-cs.cert.fr

Abstract

This paper proposes a new design approach for a secure multilevel object oriented database system. The central idea is to provide the user with a single level *virtual* database derived from an object oriented database which supports multilevel entities. This is the reason why we call this security model the Virtual View model. The database operations are performed on this virtual database within a *transaction*. The transaction ends by a *commit* which reflects back the updates on the physical object-oriented database. We argue that this approach allows us to avoid many difficulties inherent to the previous works which cope with object oriented databases supporting multilevel entities. The model we undertake in this paper can be implemented on any conventional Mandatory Security Kernel which offers to the user the possibility to perform a transaction at a level of classification chosen by the user.

1. Introduction

A problem that has been of particular interest over the last years is the management of complex and heterogeneous data (texts, maps, pictures and photographs, etc...). At present, available tools that could be used to manage these different types of information include the classical database management systems (for instance, relational or network data models) and also object oriented databases which start taking root as the new generation of database systems. Object oriented databases present important advantages on classical database systems, in particular, through the notions of encapsulation and inheritance, they are fundamentally designed to reduce the difficulty of managing complex data. They also include the notion of methods which are general programs that are associated with an object class to perform specific computation. Today, several database management systems (DBMS) based on the object oriented paradigm are already available.

One major area in DBMS is the security of shared data stored and manipulated via various operations in the database. Database security aims to maintain the confidentiality and integrity of information in the database by restricting access to authorized persons and operations only. Already now, several proposals have appeared in the literature dealing with security models for object-oriented databases. Some of them deal with Discretionary Access Controls policies (see [Pfe88], [Fer89], [Ber92] for instance). However, in the remainder of this paper, we only pay attention to Mandatory Access Controls policies, especially the multilevel confidentiality policy. In this context, most of the security models are based on the traditional object-subject paradigm of Bell and LaPadula [Bel75]. As was noticed in [Jaj90], a first problem we encounter when we want to apply the model of Bell and LaPadula to object oriented systems is that the notion of object in the object oriented model does not correspond to the Bell-LaPadula notion of object. The object oriented model combines the properties of a passive information container, represented by the attributes and their values, with the behavior of an active agent, represented by the methods and their invocation. Another problem, perhaps related to the first one, is that it is difficult to assign classifications to the items introduced in the object oriented model. Some proposals consider that every object is assigned a unique classification that applies to all its content (attributes and methods) [Mil92] or only to the passive content (attributes) [Jaj90]. The advantage of this approach is the simplicity with which security policies can be stated and enforced. However, as objects are used to model real world entities, it may seem somewhat restrictive that all objects have only a single security level. Hence, other proposals introduce also a finer grain of classification in assigning a classification to each pair (attribute, value) [Kee89, Var91]. This approach allows us to easily represent multilevel entities in the object oriented database, but researchers who are in favour of single-level objects generally consider that multilevel objects are likely to introduce overwhelming difficulties (cf. [Lun90] for instance) or try to demonstrate that to restrict objects to single-level does not necessarily imply that it would still not be possible to represent multilevel entities [Jaj90], [Bou93a].

This paper proposes a new design approach for a secure multilevel object oriented database system. It is based on an object oriented database which supports multilevel entities as in [Kee89, Var91]. However, we agree with [Lun90] and consider that supporting multilevel entities would be impractical as such. In [Bou93a], we

proposed a technique to represent multilevel entities in using single level objects. [Bou93a] may be seen as the actual implementation of the physical database supporting multilevel entities. Due to space limitation, we will not address this issue in this paper. Our approach¹ in this paper shows how to use the concept of virtual database to implement a multilevel security policy on top of the physical multilevel database. This approach drastically differs from the one proposed in [Jaj90]. It also differs from, but is actually complementary to, the security-policy proposed in [Bou93a]. When a real user decides to perform a given work, he has first to choose a current classification level which must always be dominated by the user's clearance. This user is then provided with a single level *virtual* database which is derived from the multilevel database and depends on the current classification level chosen by the user. This is the reason why we call this security model the Virtual View model. The database operations are performed on this virtual database within a *transaction* at this current classification level. The transaction ends by a *commit* which reflects back the updates on the real multilevel database. We argue that this approach allows us to avoid many difficulties encountered in previous works undertaking the design of object oriented database supporting multilevel entities.

The remainder of this paper is organized as follows. Section 2 proposes a sketch of formalization of the main elements of the object-oriented data model without dealing with security. We do not consider this model complete but it is sufficient for the purpose of this paper. Section 3 shortly reviews the concept of virtual database emphasizing those aspects which are important in the context of object oriented database security. In section 4, we propose a general overview of our approach making the assumptions that will be used in the following of this paper. Section 5 develops the Virtual View model for a secure multilevel object oriented database system. This model is mainly based on the concept of virtual database. In section 6, we suggest a possible sketch of implementation for this model. Finally, section 7 concludes the paper on further work that remains to be done.

2. Classical non secure model

This section presents an overview of object-oriented concepts. It is not our intention to give an exhaustive description here as a more detailed and a more complete account can be found elsewhere [Ban92]. However, we give sufficient details to explain how we introduce multilevel security in an object oriented database. Notice that the definitions we give, although not fully stated, are based on set-theory and denotational semantics approach. This will allow us to formalize the different models presented in this paper in an existing formal method (for instance VDM [Jon86], Z [Spi86], or B-method [Abr91]).

Let O_DB be an Object oriented database.

Definition NS 1. Computable entities in O_DB are objects. We can model an object as an injective function.

$object : object_ident \rightarrow object_state$

$object_ident$ stands for the identity of the object which is unique, and the $object_state$ is a set of attribute values.

Definition NS 2. Attribute value is a relation which associates an identifier (name of the attribute) with a value:

$Attribute_value : attribute_ident \leftrightarrow value$

To fully define $value$, notice that we can actually distinguish:

- *Primitive value.* A primitive value belongs to a predefined usual set (or type), like Natural, Real
- *Structured value.* A structured value is an identifier of another object; it can be the case when we deal with sub-objects.
- *Set value.* A set value is a set of primitive values or Structured values.

Notice that to be compatible with the object-oriented approach, we need means to state the class of which the object is instance of. In the following we consider that this can be done by adding a particular attribute (*instance-of*) whose value is a set of parent classes. Another solution would be to use a mechanism external to the object to get the set of its parent classes.

Definition NS 3. A method describes a behavior of a set of objects encapsulated in a class. It is defined by its *signature* and its *body* written in a programming language. The signature of the method is defined by its name, class, domain of each method parameter and domain of the value returned by the method.

$Method_name \times class \times \mathcal{P}(domain) \rightarrow domain$

$domain$ may be a class or a predefined type like Natural, Boolean.... Notice that if d denotes a domain, $\mathcal{P}(d)$ is also a domain.

Definition NS 4. A class is an injective function.

$class : class_ident \rightarrow class_state \times behavior$

The $class_state$ is a set of attributes, and the $behavior$ is a set of methods. $class_ident$ identifies uniquely the

1. We have already outlined a simplified form of this approach in [Bou93b].

class.

Definition NS 5. Each attribute is a relation which associates an identifier (name of the attribute) with a domain :

$$\text{Attribute} : \text{attribute_ident} \leftrightarrow \text{domain}$$

Hence, a class looks like a pattern object. Each attribute value of an object of a given class must be compatible with the domain of this attribute specified in this class.

The *schema* of O_DB is the overall view of the classes the database supports and their relationships (inheritance and specialization). For the purpose of our paper, this informal statement of the schema is sufficient.

Definition NS 6. We define O_DB as a set of objects:

$$O_DB = \mathcal{P}(\text{object}) \text{ with respect to } O_DB \text{ schema.}$$

3. An overview of the view mechanism

The concept of view was first introduced in the relational model to increase the flexibility of a database system. In the relational model, a view is simply a virtual relation derived from the real relations defined in the conceptual schema. From a theoretical point of view, this concept is based on the fact that the result obtained by computing a relational expression involving relations is always another relation. Hence, instead of evaluating this relational expression, we can consider that this expression simply defines a new virtual relation called a view.

Commercial DBMS such as Sybase [Syb87] and INGRES [Ing75] all use the view as the object of discretionary protection. Several applications of the view mechanism have also been proposed for the purpose of mandatory protection in multilevel databases, for instance the SeaView model [Den88] and the ASD-View model [Gar88].

In the context of the object-oriented model, the concept of view is rather new even though several extensions of the basic paradigm of view have already been proposed. Generally, the approaches are quite similar to what is done in the relational framework. However, object-oriented views are intended not only to define new data structures, as is the case in the relational model, but also new data behaviors, since the methods are an integral part of the class definition in the object-oriented model. Hence, defining new virtual classes is more complicated than defining virtual relations.

The objective of this paper is to show how to use a view mechanism to cope with some problematical aspects of multilevel object oriented database systems. The view mechanism we describe is partly inspired from the one proposed in [Abi93] for a context where security problems are not relevant. In [Abi93], views are defined as virtual schemas, from which virtual databases are derived. This means that the activation of view will result in the creation of a virtual database. Views are evaluated dynamically. This is important for performance and avoids virtual databases to be evaluated on the whole database. In [Abi93], it is also possible to associate a view with new data behaviors, i.e. new methods. However, we need not use this possibility in our approach.

4. Assumptions of our approach

Our objective is to propose an approach which allows us to deal with multilevel entities in an object oriented database without some inconveniencie encountered by the previous proposals. It is based on the derivation of single-level virtual databases from a multilevel database. All the operations are then performed on the virtual database within a transaction at a current classification level chosen by the user.

For the sake of simplicity of the exposure, we make several hypotheses. Let us first explain how the multilevel security policy is introduced within the physical database. For instance, let us consider the following class Person and one object O1 which have been created in a context where security problems are not relevant. We also assume that O1 is an instance of Person.

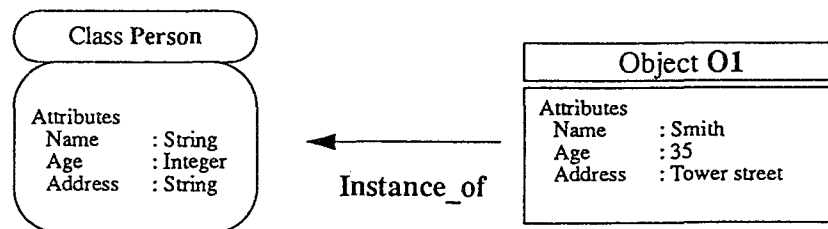


Fig. 1 Non secure database

In our approach, security levels are used to transform this class and its related instance. For instance, we may get the figure 2.

Notice that in the Virtual View model the assignment of security levels is only done at the object level (instances). We assume that the database schema is not protected and each user, whatever his clearance, can have access to the overall classes of the database. This assumption explains why the class Person is not modified in the above example. We guess that a complete model for multilevel object oriented databases should also provide the possibility to hide some part of the database schema, for instance the existence of a class or the existence of a given attribute in a class definition. This clearly represents further work that remains to be done.

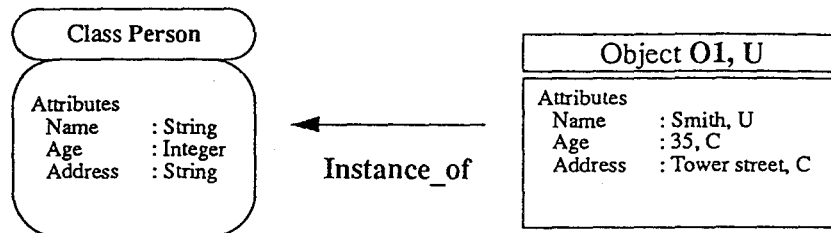


Fig. 2 Multilevel database

Notice also that the instance *O1* has a security level associated with its identifier and another for each of its attributes. The object-identifier classification is used to hide the existence of the object to the subjects which are not sufficiently cleared. In the above example, this security level is equal to Unclassified. This means that everybody is permitted to know that *O1* has been created in the database. On the other hand, if this security level had been assigned to Confidential, then subjects cleared to Unclassified would have ignored the existence of *O1*. Notice that the object-identifier classification represents the security level at which the object has been created.

We also assume that the object-identifier of a newly created object is taken in a predefined set of object-identifiers. This set is partitioned into several pairwise disjoint subsets associated with each security level. This means that each object-identifier is assigned a predefined level of classification. This level of classification is static and independent from the fact that an object associated with this object-identifier exists or not. However, when a user wants to create an object at a given security level, then this new object must be associated with an object identifier having the appropriate level of classification. This constraint is used to avoid that two objects with different security levels are identified by the same object identifier.

The attribute classification classifies the information represented by the association of the attribute with the object identifier. For instance, the value of the attribute Age is associated with the level Confidential. This means that the information "the age of *O1* is equal to 35" is Confidential. Each attribute can be labelled independently. This principle allows us to represent multilevel entities in the model.

The object-identifier must always be assigned a classification which is dominated by the greatest lower bound of the attribute classifications of the object. With this integrity constraint, a subject must be first authorized to observe the existence of the object before being authorized to access one of its attribute values.

It is also important to notice that we assume that the attribute classification itself is classified at the same level of classification as the object-identifier classification. This means that, if a user is permitted to know the existence of an object, then he is also permitted to know the security level of any attribute value. It implies that the existence of a sensitive attribute value is not directly protected. Therefore, we do not need to use polyinstantiation to introduce a cover story whose purpose is to hide the existence of an otherwise sensitive attribute value¹. When an unclassified user queries the database to know the value of a sensitive information, he may be told that this information exists but is sensitive without generating an unauthorized flow of information through a covert channel. For this purpose, we introduce special values called "level values". Formally, each security level *l* is associated with a level value denoted "*l*" and, if an unclassified user wants to know the value of a sensitive piece of information classified at level *l*, then this user is provided with the special level value "*l*"².

5. Security model

In this section, we develop the Virtual View model for a multilevel object-oriented database system using the assumptions of section 4. We begin, in section 5.1, by modifying the classical non-secure model proposed in section 2, in order to obtain a model for our real multilevel object oriented database. Then, section 5.2 shows how to model a single-level virtual database and section 5.3 develops the rules to derive this virtual database from the real multilevel database of section 5.1. In section 5.4, we describe the commit which is run at the end of a transaction. Section 5.5 illustrates our approach through an example and sections 5.6 and 5.7 respectively deal with object creation and method activation. Finally, the difficulty of implementation is considered in section 6.

1. A cover story is a lie introduced by the high users to cause the low users to believe something that is incorrect.

2. This is a generalization of the special symbol *restricted* used in [San92] to tell users that corresponding values exist but are higher classified. Our special "level" values provide the user with more precise information.

5.1 Multilevel Entities

In the real object-oriented database model, attribute values are assigned classification levels. Hence, an attribute value may be unclassified, confidential, secret and so on depending on the choice made by the creator of the correspondant object. Consequently definition NS1 and definition NS2 are slightly modified to handle these security levels.

Definition ML 1. An object is defined as an injective function which associates the identity of the object with an *object_state*.

$$\text{object} : \text{object_ident} \rightarrow \text{object_state}.$$

Moreover, a sensitivity is assigned to the object :

$$\text{sensitivity} : \text{object_ident} \rightarrow \text{level}$$

The purpose of the *sensitivity* function is to classify the object existence.

As in the non-secure model, *object_state* is a set of attribute values. However, the definition of attribute value is changed as follows:

Definition ML 2. Attribute value is a relation which associates an identifier (name of the attribute) with a value:

$$\text{Attribute_value} : \text{attribute_ident} \leftrightarrow \text{value}$$

Moreover a sensitivity is assigned to the association between an object identifier and the *Attribute_value*

$$\text{sensitivity} : \text{object_ident} \times \text{Attribute_value} \rightarrow \text{level}$$

value may be a value as defined in the definition NS 2. Moreover the following integrity constraint must be enforced:

- $\forall o : \text{object_ident}, \forall va : \text{Attribute_value}, va \in \text{object}(o) \rightarrow \text{sensitivity}(o, va) \geq \text{sensitivity}(o)$

5.2 Virtual single level database

We could directly define a security policy for the multilevel database. This policy could be closely related to the one proposed for the SODA model [Kee89] by example, but such a policy would present a few drawbacks (see [Bou93a] for a more detailed discussion of the SODA model):

- Every time a read or write occurs, a security level control would have to be effected, possibly causing a performance loss.
- Compared with the case of a non secure database, administration of methods would have to be adapted because we would need to assign a current security level to a running method.
- Multilevel transactions would introduce timing channels.

Our objective is to propose an alternative approach. For this purpose, we can create from a multilevel database as many virtual databases as security levels. Each virtual database is classified by a unique security level. Thus, all objects and all attribute values inside the virtual database are implicitly classified with the same security level.

When a user is to perform a given work, he has first to choose a current classification level. Of course this working level is dominated by his clearance. Then, a single level transaction, assigned with this current classification level, is initiated and the user is provided with a virtual single level database which is derived from the multilevel database. The security level of this virtual database is the same as the current working level.

Once this virtual database is created, the user may access it within the single level transaction. He may read and update attribute values without any restrictions or access controls¹. At the end of the transaction, the user may order a commitment which reflects back the updates performed on attribute values assigned with a security level which is the same as the current working level. Access controls to propagate updates are effected when the commitment is complete.

In virtual database, we no longer need to classify attribute values and objects. Instead, we assign a security level to the whole virtual database. Thus, compared with the classical non secure model, only definitions NS 2 and NS 6 are modified into defintions VV 2 and VV 6.

Definition VV 2. Attribute value is a relation which associates an identifier (name of the attribute) with a value:

$$\text{Attribute_value} : \text{attribute_ident} \leftrightarrow \text{value}$$

Value may be a value as defined in the definition NS 2 or may be taken into the set of "level" values which are defined as follows:

1. Notice that [Atk89] says that in object-oriented database, there are cases where encapsulation is not needed. In our model, encapsulation has no influence on security. The user may directly read or write attribute values without invoking a method. Other works are based on strict encapsulation and require that a message filter can intercept every message sent by an object in the system [Jaj90].

- "Level" value. Each security level l is associated with a level value denoted " l "

Providing the user cleared at level l_1 with a level value equal to " l_2 " means that the value exists but is classified at level l_2 . This only makes sense when the user clearance l_1 is not higher than l_2 . Indeed, if l_1 is higher than l_2 , then this user would be provided with the actual value of the information. Notice that we assume these "level" values belong to every domain.

Definition VV 6. We define O_VDB as a set of objects:

$O_VDB = \mathcal{P}(\text{object})$ with respect to O_VDB schema.

Moreover, a sensitivity is assigned to the whole virtual database:

$\text{sensitivity} : O_VDB \rightarrow \text{level}$

5.3 Rules to derive a virtual database.

We now describe the rules which are needed to obtain a virtual database from a real multilevel database:

- **Rule 1.** *As the schema is unclassified, it is propagated as such in any virtual database.*

Once the view, i.e virtual schema, is created, virtual objects are evaluated as follows:

- **Rule 2.** *If an object has a security level which is not dominated by the security level of the virtual database, then this object is not propagated in the virtual database.*

Notice that such a propagation would be prevented by the Mandatory Security Kernel.

- **Rule 3.** *If an object has a security level dominated by the security level of the virtual database, the object may be propagated in the virtual database.*

As is noticed in [Abi93], a view may be evaluated dynamically. Virtual instances are created only if a user or process wants to perform on it.

- **Rule 4.** *If an attribute value of an object has a security level dominated by the security level of the virtual database, the attribute value may be propagated as such in the corresponding virtual object.*

- **Rule 5.** *If an attribute value of an object has a security level l_i which is not dominated by the security level of the virtual database, the attribute value is enforced to " l_i "¹ in the corresponding virtual object. This is to avoid a read up.*

Notice that such a read up would be prevented by the Mandatory Security Kernel. Once a transaction has been initiated and a virtual database has been created, the user or a method can read or write attribute values in this virtual database without any controls and any restrictions. It is important to stress that as long as the user or the process does not perform a commitment, updates are not propagated in the physical database.

5.4 Commitment.

At the end of a transaction, user or process may order a commitment. The main principle of this commitment is to propagate updates only for attribute values which are at the same level as the virtual database level. More precisely the rules are expressed as follows. For every attribute value in the virtual database,

- **Rule 6.** *If the level of the corresponding attribute value in the multilevel database is not higher than the level of the virtual database then the update is not propagated. This is to avoid a write down.*

Notice that the commitment would not need to be trusted because such a write down would be prevented by the Mandatory Security Kernel.

- **Rule 7.** *If the level of the corresponding attribute value in the multilevel database is equal to the level of the virtual database then the update is propagated.*

- **Rule 8.** *If the level of corresponding attribute value in the multilevel database is higher than the level of the virtual database then the update is not propagated. This is to avoid a write up. It is not a confidentiality requirement but an integrity one.*

5.5 Animation of the model.

We give a small database which is sufficient to illustrate our purpose. There are two classes (Person and Employee) and two objects ($O1$ and $O2$). With the intent to simplify our example, we give only three security levels {Unclassified, Confidential, Secret}. Figure 3 shows the physical multilevel database. We mention in each class only specified attributes and specified methods.

Figure 4 shows the unclassified virtual database. Only $O1$ which is unclassified is propagated in the virtual

1. As we have assumed that the attribute classification is classified at the same level of classification as the object existence classification, the user is permitted to observe this "level" value. We will show in section 5.6 how this assumption may be made without creating a covert channel.

database. Every confidential or secret attribute values of *O1* are enforced to "Confidential" or "Secret" in the virtual database. At the end of the transaction, commit propagates updates for unclassified attribute values only. Since it is unclassified, this database is accessible by any user or any method underlying a user.

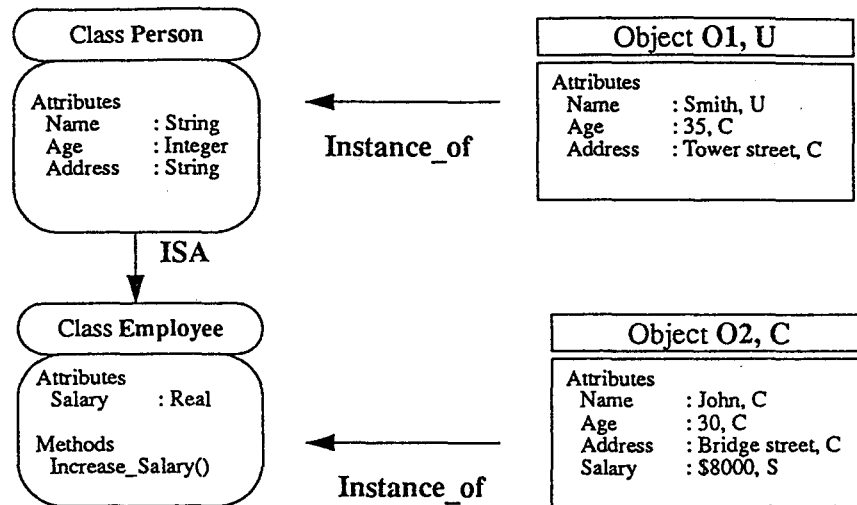


Fig. 3 Multilevel database

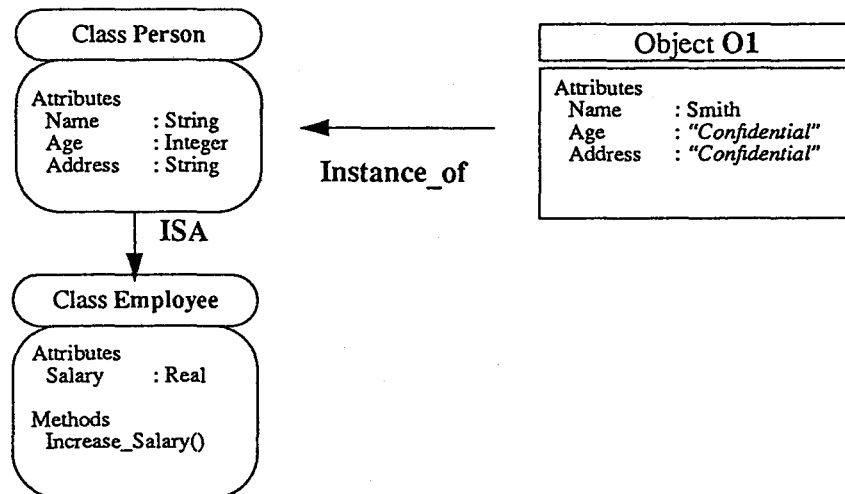


Fig. 4 Virtual unclassified database

Figure 5 shows the confidential virtual database. Both *O1* and *O2* are propagated in the virtual database. Every secret attribute value is enforced to "Secret" in the virtual database. At the end of the transaction, commit propagates updates for confidential attribute values only. Since it is confidential, this database is accessible by users with a clearance equal to the confidential or the secret level.

Secret virtual database is quite similar to the confidential virtual database. The only difference is that no attribute value is hidden by a "level" value. In particular the salary of *O2* is equal to \$8000. Of course, this database is accessible by secret users only. During commitment, updates are propagated for secret values only.

5.6 Object creation

Object creation may be performed in any virtual database but the following rule must be enforced:

- **Rule 9.** The object-identifier of a newly created object must be taken in the predefined subset of object-identifiers assigned with a security level which is the same as the security level of the current virtual database.

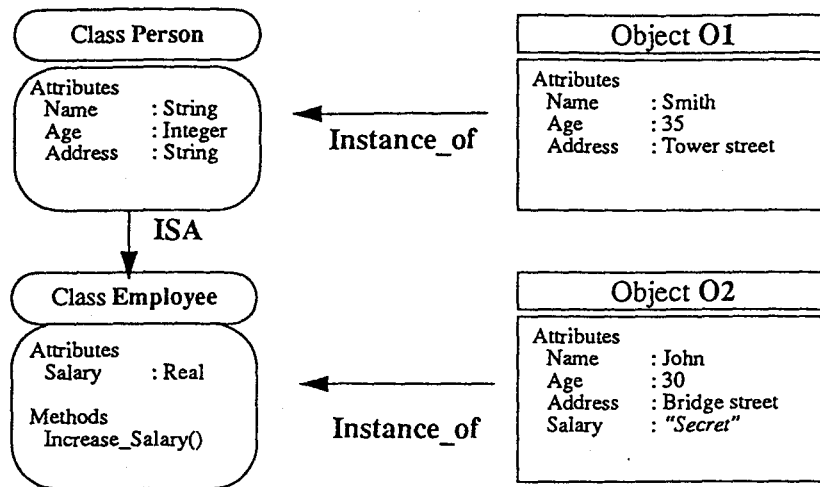


Fig. 5 Virtual confidential database

This entails that the classification of the object existence be the same as the security level of the virtual database in which the object is created. Without this rule, a method trapped by a Trojan horse could use object creation as a way to disclose some information. The object creator is also in charge of associating the attribute values of the newly created object with sensitivities. As a user in the virtual database does not directly deal with security levels, assigning sensitivities to the attribute values is indirectly done as follows. Once the object creator has got an identifier, he may assign attribute values to the object. These attribute values may be "level" values or "normal" values¹. Of course the following rule must be enforced:

- **Rule 10.** *If an attribute value is associated with a level value " l_i ", then the corresponding security level l_i of the level value " l_i " must dominate the current security level. This is to respect the integrity rule defined in definition ML 2.*

After commitment, the object is created in the physical database. The attribute values assigned to the object in the virtual database are propagated in the physical database. There are two different cases of propagation:

1. If, in the virtual database, the object creator has associated an attribute with a "level" value " l_i ", then, in the physical database, this attribute is associated with a "Null" value whose sensitivity is equal to l_i
2. If, in the virtual database, the object creator has associated an attribute with a "normal" value, then, in the physical database, this attribute is associated with this "normal" value whose sensitivity is equal to the security level of the current virtual database.

These two rules imply that the classification of the security levels assigned to the attribute values of the newly created object will be the same as the classification of the object existence (i.e. the security level of the current virtual database). This means that in knowing the object existence, a user may also observe the security levels associated with each attribute value of this object.

To fully instantiate the new object, the user must then successively set himself to each security level which appears in attribute values of the object. Then he may successively and normally update "Null" values with "normal" values via virtual databases. Due to rule 7, the user may only update a level value " l_i " by a "normal" value if the current level of the virtual database is actually equal to l_i . Of course, to fully instantiate an object, user's clearance must dominate the least upper bound of security levels which appear in the attribute values of the object. Let us see in figure 6 how a confidential instance of the class Person would be created by a secret user. This figure shows creation of a new confidential object O3 in the class Person (we mention only the newly created object and the class Person). Classifications assigned to the attribute values of the object O3 will be confidential for the Name, confidential for the Age and secret for the Address. At first, secret user (or process) requesting the creation must set his working level to confidential. He may invoke the primitive of object creation and assign "normal" values to the Name and Age and a "Secret" value to the Address. In the physical database, after commitment, a confidential multilevel object is created. The "normal" values of the Name and Age are propagated in the physical database and the attribute value of the Address is enforced to Null. Then, the user may raise his working level to secret to fully instantiate the new object by updating the Address attribute value.

1. By "normal" value, we mean any value which is not a "level" value.

5.7 Method activation

Another advantage of the Virtual View model is that compared to a non secure database, administration of methods does not need to be modified. Indeed, as methods are executed in a single level virtual database, they may read or write attribute values without security controls. In other words, applying information flow controls to method invocation would require trusted information mechanism embedded in the object layer [Kee89, Jaj90]. In particular, our approach does not require to implement mechanisms to allow an object to communicate with another object of a different level. Such mechanisms are tedious to implement without generating timing channels (see [San91] for a discussion).

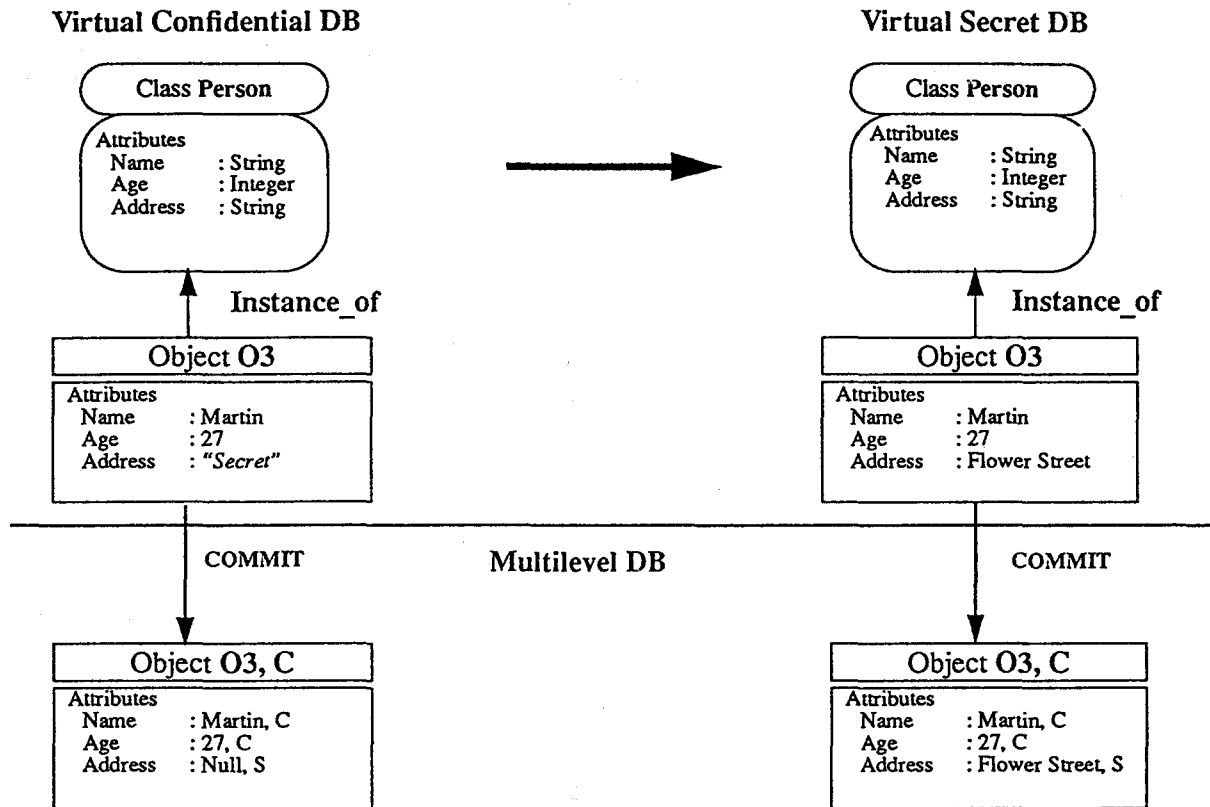


Fig. 6 Confidential object creation

6. Sketch of implementation

In this section, we briefly consider the difficulty we may encounter when we implement the Virtual View model. First, we argue that our proposal for a secure multilevel object oriented database system can be implemented on any conventional Mandatory Security Kernel which offers to the user the possibility to perform a transaction at a level of classification chosen by the user. In this case, the architecture of the system may be described by Figure 7.

Through this figure, we can see how a transaction is performed and the different entities at work. The starting point is a user who wants to query the database. First, this user, has to choose a current level of classification he wants to associate to his query. The Mandatory Security Kernel performs several security controls in particular the authentication of this user. Then, the transaction begins. A virtual database is provided to the user. It is built from a view of the multilevel database and depends on the level of classification chosen by the user. This virtual database is built by a specific (non standard) part of the DBMS called the Virtual Database Extractor. When the virtual database is generated, security controls are performed by the kernel. This kernel must enforce that the virtual DB extractor only accesses physical databases having an appropriate classification with respect to the current level of the transaction (Physical Access Control) and that the virtual database is stored in a memory area having an appropriate classification (Memory Access Control). Notice that the structure of this virtual database is almost identical to the structure of a non secure database; the only modification is the use of specific "level" values. Hence, we guess that the query may be handled by a standard DBMS. Finally, a commit is done at the end of the transaction. This commit is also non standard and requires some modification of the commit existing in a standard DBMS. The Mandatory Security Kernel checks all the operations performed by this non standard commit.

We can consider that the global multilevel DBMS is actually a modified standard DBMS with two specific functions: the first one is the Virtual Database Extractor and the second is the Commit. Hence, it seems that this approach might be implemented with only minor modification of an existing standard object oriented database. In particular, notice that Virtual Database Extractor and Commit need not be trusted components. Indeed, we may consider that the Mandatory Security Kernel performs all the security controls. However, if we consider performance issues, it may be interesting to insert them in the security kernel to avoid unnecessary controls which would otherwise be performed by a classical security kernel. Notice also that a Standard Security Kernel can generally enforce mandatory controls with respect to single level information containers. Hence, a Standard Security Kernel would not allow to manage multilevel objects as such. It is necessary to decompose multilevel objects into single level objects which are then physically stored in the database. Due to space limitation, we cannot describe the decomposition mechanism in this paper but we refer to [Bou93a] for a detailed presentation. This decomposition mechanism called MultiView is fully compatible with the use of Virtual Database described in this paper.

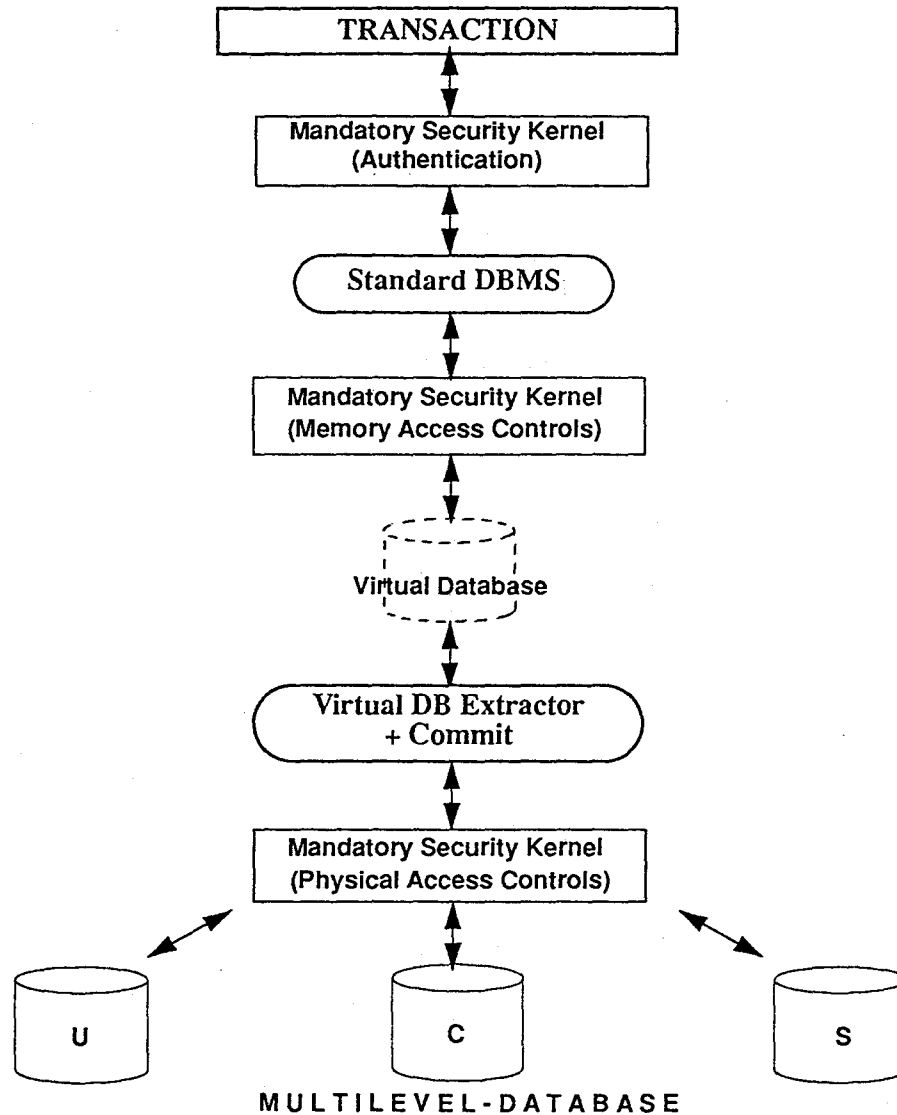


Fig. 7 Implementation

Finally, notice that as in the SODA model a covert timing channel may exist in our model. It arises when a user starts a new high level transaction and then returns to the unclassified level after a commit. The transaction can read high level data and the commit can communicate it by modulating its execution time. Several solutions to avoid this type of timing channel exist, for instance we may require that the commit operation only be executed under the control of the user through a trusted path.

We plan to develop a prototype to explore the feasibility of the Virtual View model and to combine it with the MultiView model described in [Bou93a]. This would allow us to study if additional complexity that was not

discovered during the conception phase may arise during the implementation phase.

7. Conclusion

Our objective in this paper was to show how to fruitfully use the concept of virtual database to avoid many difficulties which arise when we deal with multilevel entities in an object oriented database. The Virtual View model presents several advantages. First, it can rely on an underlying security kernel for the enforcement of mandatory security properties. This means that the components providing the virtual object oriented database need not be trusted, while other works require trusted enforcement mechanisms in the object layer [Kee89,Jaj90]. We also feel that our approach requires only minor modification of an existing object oriented DBMS. This is another advantage of the Virtual View model which allows us to deal with single level database. It was not the purpose of this paper to address performance issues. However, in this field, our approach may also present several advantages because, after the virtual single level database has been generated, it does not require any security control. Hence, the analysis of the performance could reveal significant gains.

This preliminary work could be extended in several directions. A first extension consists in classifying some part of the object oriented database in order to hide some part of this schema, for instance the existence of a secret attribute. We have also noticed that we do not need to use polyinstantiation in our model, but, as a drawback, it would not be possible to support cover stories. Hence, another extension would be to propose means to support cover stories to hide the existence of a secret value of an attribute. It is generally considered that polyinstantiation is well adapted for this purpose, but we must investigate if other solutions do not exist in the context of object oriented databases.

Finally, in [Bou93a], we have suggested another approach based on the decomposition of an object oriented database which supports multilevel entities in a collection of single level databases. We guess that the approaches described in this paper and in [Bou93a] are fully compatible and complementary. However, how to exactly combine them requires further investigation. Notice in particular that the approach described in [Bou93a] already provides the possibility to support cover stories.

References :

- [Abi93] S. Abiteboul, Cassio Souza dos Santos and C. Delobel. *Virtual Schemas and Bases for dynamic Data-Intensive Systems. To be published at the next EDBT 94 conference. Cambridge. England.*
- [Abr91] J.R. Abrial. *The b method for large software specification, design and coding (abstract). In Prehn Toetenel, editor, VDM'91, vol 2. Springer Verlag, 1991.*
- [Atk89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier and S. Zdonik. *The Object-Oriented Database System Manifesto. Proc. of the first international conference on Deductive and Object-Oriented Databases (DOOD89).*
- [Ban92] F. Bancilhon, C. Delobel and P. Kanellakis. *Building an Object-Oriented Database System. Morgan Kaufmann 1992.*
- [Bel75] D. Bell and L. Lapadula. *Secure Computer Systems : Unified Exposition and Multics Interpretation. Technical report, MTR-2997, MITRE, Bedford, Mass, 1975.*
- [Ber92] E. Bertino. *Data Hiding and Security in an Object-Oriented Database System" Proc. Eight IEEE International Conference on Data Engineering, Phoenix, Arizona, 92.*
- [Bou93a] N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, K. Yazdanian. *Multiview Model for MultiLevel Object-Oriented Database. To be presented at the Ninth Annual Computer Security Applications Conference. December 93. Orlando, Florida.*
- [Bou93b] N. Boulahia-Cuppens, F. Cuppens, A. Gabillon, K. Yazdanian. *Techniques to Handle Multilevel Objects in secure Object-Oriented Databases. Proc of the OOPSLA 93 Conference Workshop on Security in Object-Oriented Systems. Washington D.C., USA. September 93.*
- [Den88] D.E. Denning, T.F. Lunt, R.R. Schell, W.R. Shockley and M. Heckman. *The Sea View Security Model. Proc. of the 1988 IEEE Symposium on Research in Security and Privacy. Oakland.*
- [Fer89] E. Fernandez, E. Gudes, H. Song. *A Security Model for Object-Oriented Databases. Proc IEEE Symp. on Security and Privacy, Oakland 89.*
- [Gar88] C. Garvey and A. Wu. *ASD-Views. Proc. of the 1988 IEEE Symposium on Research in Security and Privacy. Oakland.*
- [Ing75] G. Held, M. Stonebreaker and E. Wong. *INGRES - A Relational Database System National Computer Conference 1975.*
- [Jaj90] S. Jajodia and B. Kogan. *Integrating an object-oriented data model with multi-level security. Proc. of the 1990 IEEE Symposium on Security and Privacy.*
- [Jon86] C.B. Jones. *Systematic Software Development using VDM. Prentice-Hall International, Inc., 1986.*
- [Kee89] T.F. Keefe, W.T. Tsai and M.B. Thuraisingham. *SODA : A Secure Object-Oriented Database System. Computer & Security, Vol 8, N°6, 1989.*
- [Lun90] T.F. Lunt. *Multilevel Security for Object-Oriented Database Systems. Database Security III : Status and Prospects. Elsevier Science Publisher B.V. (North-Holland). IFIP 1990.*
- [Mil92] J.K. Millen and T.F. Lunt. *Security for Object-Oriented Database Systems. Proc. of the 1992 IEEE Symposium on Research in Security and Privacy.*
- [Pfe88] H. Pfeifferle, M. Härtig, K. Dittrich. *Discretionary Access Control in Structurally Object-Oriented Database Systems. Proc. IFIP WG 11.3 Workshop on Database Security, Ontario, Canada, 88.*
- [San91] R. Sandhu, R. Thomas, S. Jajodia. *Supporting Timing-Channel Free Computations in Multilevel Secure Object-Oriented Databases. Database Security V. Status and Prospects. C.E. Landwehr, S. Jajodia Editors. North Holland.*
- [San92] R. Sandhu and S. Jajodia. *Polyinstantiation for cover stories. Computer security - Esorics 92. Springer Verlag.*
- [Spi86] J.M. Spivey. *The Z Library. A reference manual, Programming Research Group, Oxford University, 1986.*
- [Syb87] *Sybase User's Guide, Sybase Inc, 1987.*
- [Var91] V. Varadharajan and S. Black. *Multilevel Security in a Distributed Object-Oriented System. Computer & Security, 10 (1991).*

ACHIEVING DATABASE SECURITY THROUGH DATA REPLICATION: THE SINTRA PROTOTYPE

Myong H. Kang, Judith N. Froscher, John McDermott, Oliver Costich, and Rodney Peyton

Naval Research Laboratory
Information Technology Division
Washington, D.C. 20375

Abstract

There are several proposed approaches for multilevel secure (MLS) database systems which protect classified information. The SINTRA¹ database system, which is currently being prototyped at the Naval Research Laboratory, is a multilevel trusted database system based on a replicated data approach. This approach uses physical separation of classified data as a protection measure. Each database contains data at a given security level and replicas of all data at lower security levels. Project goals include good performance and full database capability.

For practical reasons (e.g., ease of evaluation, portability) the SINTRA database system uses as many readily-available commercial components as possible. In this paper, security constraints and the rationale for the SINTRA prototype are described. We also present the structure and function of each component of the SINTRA prototype: the global scheduler, the query preprocessor, and the user interface. A brief description of the SINTRA recovery mechanism is also presented.

1 Introduction

As government downsizes, there is a growing need to exploit the sizable commercial investment in information technology to carry out government responsibilities more efficiently and effectively. With greater reliance on information systems, both government and commercial organizations are vulnerable to attacks on the confidentiality, integrity, and availability of information. For more than a decade the government has

supported research in computer, communication, and information security to protect against such threats. The results of this research can provide the basis for managing data securely in the new information infrastructure.

In this paper, we describe the SINTRA prototype database management system. SINTRA enforces a strong protection policy and provides both good performance and the full data management capability of conventional industry-standard database management systems. The Naval Research Laboratory is developing the SINTRA prototype to demonstrate the feasibility of using physical separation and replication as the primary protection mechanisms for a database system that provides both high assurance and multilevel security.

First, we provide a brief summary of computer and database security concerns and a survey of several possible approaches to database security. Next, we present an overview of the SINTRA prototype architecture. A model of the security constraints enforced by the prototype allows consideration of how these constraints affect the transaction model and data model for the prototype. We then describe the process structure of the prototype SINTRA scheduler and a recovery strategy for this approach. The preprocessor manages the security labels as data in the conventional relational data model and ensures that proper modifications are made to user commands. Even though the SINTRA prototype is really a MultiLevel Secure (MLS) database server, a user interface has been developed that illustrates how labeled data can be shown to the user. We conclude with descriptions of the SINTRA prototype status, what we have learned, and future questions that the prototype must address.

¹Secure Information Through Replicated Architecture

2 Background

Organizations have long known the importance of restricting access to sensitive information to prevent competitors from learning about plans, new products, or changes in strategies. One approach for controlling access to this information is to allocate the information to sensitivity classes and restrict access based on the consequences of the information's compromise. This approach also requires that those who must access sensitive information be assigned to authorization classes commensurate with the sensitivity level of the information they are allowed to access and their trustworthiness, which is assessed through a background investigation.

When the government first used computers to process sensitive information, all users had to be cleared for access to the highest level information processed. The logical extension of this policy would have resulted in all workers being cleared for the most sensitive information. Because the government has not cleared everyone to the highest level, it has been forced to use some relatively insecure approaches to sharing information among users with different clearances.

Trusted computer products must be evaluated and tested in an adversarial manner. For high-assurance systems that must provide strong separation, the protection mechanisms must be simple and easy to evaluate. Protection critical components must contain only protection mechanisms and must mediate every access by each user. These systems are more highly engineered and crafted than most computer systems. Protection critical components must be scrutinized in an adversarial way that ensures that neither malicious code nor covert channels have been introduced into the system.

When high-assurance application systems are developed, system engineers must take great care to design systems that take advantage of the strong separation provided by these products but rely on them for little else. The commonly accepted theory for developing MLS systems is to develop untrusted applications that run at a single level but can access all data at that level and below. If the application is data intensive or requires communication across security levels, achieving both high assurance and good performance is unusually difficult. The thing to be avoided at all cost is changing the protection critical part of an already-evaluated high-assurance component. Any change invalidates the evaluation. Now we can examine the various approaches that have been taken to building a

high assurance, MLS database management system.

The most straightforward approach to providing multilevel security in a database system is to design the security mechanisms into the database system itself and trust the database to enforce the security policy. In practice this results in a low assurance system, that is, the separation is weak. This kind of database system is useful, but cannot interconnect the diverse population of users expected for the new information infrastructure. The reason for the low-assurance is the complexity and size of modern database systems.

Less straightforward but more effective approaches use a reference monitor to enforce the security policy. The reference monitor is evaluated for high assurance and the database system is designed to function under the security constraints enforced by the reference monitor.

Following this reasoning, the Multilevel Data Management Security Summer Study [Air83] recommended three near-term approaches to solving the multilevel database security problem. The three approaches are: integrity lock, kernelized, and distributed. Within the latter there are two sub-approaches: replicated and non-replicated.

The integrity lock approach [Den85] uses a trusted frontend, a single untrusted backend DataBase System (DBS), and encryption techniques to protect data. A trusted frontend applies an encrypted checksum to data stored by an untrusted backend database system. The integrity lock approach is computationally intensive because checksums have to be computed whenever data are inserted or retrieved. Since the trust is in the frontend filter and the backend DBS stores data from multiple security levels, this architecture is susceptible to Trojan horse attack. Hence this approach cannot be used for highly-assured MLS DBS (e.g., a B3 or A1 system [DoD85]).

The kernelized approach [Lun90], relies on decomposing the multilevel database into single-level files which are stored separately under the control of a security kernel enforcing a mandatory access control (MAC) policy. Separate untrusted DBS are run at each security level. Decomposing multilevel relations into single-level relations so that the recomposition of the fragments is the same as the user's view of the multilevel relation has, in every case, presented many challenges. There are also problems in preserving full database functionality (e.g., transaction management, data model) while providing the required security. Materializing multilevel relations from single-level base relations requires fairly complex mechanisms and may

degrade performance. In addition, the need to do this materialization has forced limitations on the data model (e.g., uniform classification of keys [Lun90]) and results in difficulties in representing *many-to-many* relationship [KCF93b]. Since this approach uses a trusted operating system to enforce separation of data at different security levels, the security of this architecture is as strong, in theory, as the security of the trusted OS.

The distributed approach comprised two architectural sub-approaches (1) each DBS has data at a single security level (non-replicated approach), and (2) each DBS contains data at a given security level and replicas of all data at lower security levels (replicated approach).

The non-replicated approach has been investigated by Jensen *et. al.* [Jen89, OcG88]. This architecture has a trusted frontend and many untrusted backends which may be commercial DBSs. Each backend contains data of only a single security level. This approach has inherent security problems because higher level queries have to be propagated to lower level untrusted backends to request data. Sending requests down to lower security levels introduces a covert channel, which can be used to transmit information from higher level backends to lower level backends. Hence, we don't expect that this approach can be used for highly-assured MLS DBS. This approach also can yield reduced performance because it may require fragments to be transferred from a low backend DBS to a high one in order to present a multilevel relation to users.

The replicated approach uses physical separation as a protection measure. A Trusted Front End (TFE) mediates access to separate Untrusted Backend DBSs (UBD) for each security class. Each backend DBS contains information at a given class and replicated information from all lower backend databases. Hence, all the information that a user can legitimately view is located at the backend corresponding to the user's authorization.

3 Description of The SINTRA System

Security, performance, and portability concerns led NRL to the initiation of a project to investigate replication using commercial DBS as a promising alternative for building a MLS DBS. Goals include good performance and full database functionality.

3.1 Practical Considerations

Security is an important issue for many organizations. However, there are other important issues, such as the reduction of production, operational, and maintenance costs and minimizing development time, effort, and evaluation costs. The ability to integrate new technology into high assurance, MLS systems in a straightforward, timely fashion makes the replicated approach very attractive to cost conscious decision makers.

One way to achieve these goals is to make maximum use of existing products which are already tested and evaluated. Those products are usually maintained by the vendors' staffs. For example, special purpose computer systems can be built by connecting general purpose commercial products together. Many interface standardization efforts (e.g., the OSI standard) make this approach more viable. Developing a new product from commercial subcomponents has several advantages:

- Minimal development and maintenance costs.
- Easy to upgrade. Once better and cheaper products are on the market, they can be easily incorporated.
- Easy testing and evaluation. Most of the subcomponents, which are commercial products, have been evaluated.
- Easy to connect. Since each subcomponent may already conform to interface standards, it is likely that the new product will be easy to interface to other products.

Based on the practical considerations above, the SINTRA prototype uses as many commercial components as possible, resulting in the need for relatively little new work to construct the overall MLS system. In the remainder of this paper, we show how the SINTRA project achieves the practical goals that we specified.

3.2 Overview

The SINTRA database system consists of one trusted front end (TFE) and several untrusted backend database systems (UBD). The role of the TFE includes authenticating users, directing user queries to the proper backend, maintaining data consistency among backends. The UBD at each security level contains data at its own security level, and replicated data from lower security levels.

The SINTRA database system prototype at the Naval Research Laboratory uses a Honeywell XTS-200 system, which is a high assurance trusted OS (B3 rated system), as a trusted frontend. Since each UBD is treated as a "black box" (i.e., inputs and outputs of the system are known, but its internal behavior is not known), each UBD can be any commercial database system. Currently, untrusted ORACLE 7 database running on SUN4/300 is used as each backend database. The backend and frontend computers are connected through dedicated Ethernet connections. Since the SINTRA security policy is enforced by the frontend, the security of this architecture is as strong as the security of the trusted frontend. Hence, the SINTRA system will be a B3 MLS DBS. Figure 1 illustrates the SINTRA architecture.

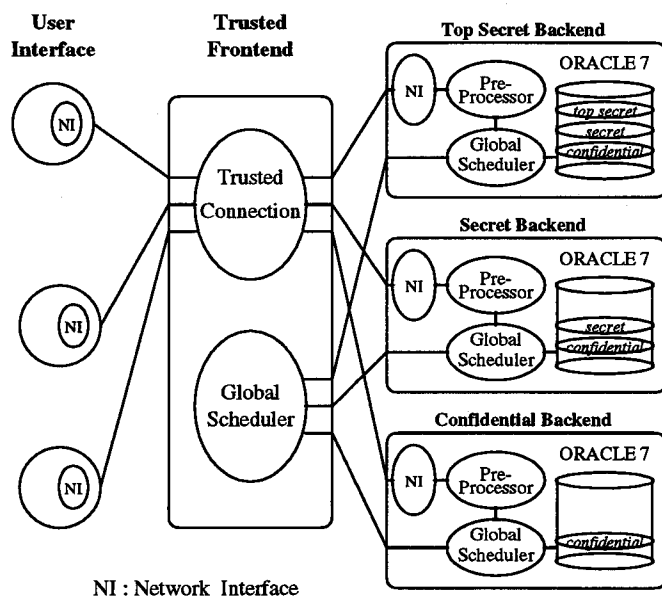


Figure 1: The SINTRA Architecture.

When a user attempts to login to the SINTRA prototype, the trusted connection checks the user's security level and establishes a connection to the network interface at the backend for that security level. Now, the user is ready to issue commands. A user query which is received at the backend will be modified by the query preprocessor, if necessary. A modified query will be submitted to the ORACLE database system through the global scheduler. The responsibilities and the detailed description of these components will appear in sections 5, 6, 7, and 8.

The SINTRA prototype is based on the following:

- All UBDs use the same database query language (e.g., SQL).
- The TFE changes the database states of the UBD only through database queries.
- Each UBD performs transaction management. (If it produces serializable and recoverable histories, SINTRA will as well.)

Note that the above assumptions enable us to treat each UBD as a black box that operates autonomously without the knowledge of other UBD's or global scheduler. Hence, no modification of the commercial DBS or the commercial frontend is required. However, we cannot expect to obtain the desired security and functionality by simply connecting commercial products. Custom processes, such as query preprocessor and global scheduler, ensure the delivery of desired database security and functionality.

The SINTRA approach has several advantages.

1. Data retrieval performance is better because the SINTRA system does not materialize the user's view from single-level relations (as do the kernelized and non-replicated distributed architectures)².
2. Mandatory access control is enforced through physical separation of data. Since SINTRA uses an evaluated product as a trusted frontend, very little trusted code needs to be developed to ensure that this separation is maintained.
3. Development and maintenance cost can be reduced because commercial frontend and backend database systems are widely available.
4. Performance can be improved by using optimization and parallelization techniques that have been developed for conventional databases, because the replicated architecture uses conventional database systems as UBDs. Uniprocessor or multiprocessor computers can be chosen as backend computers without affecting the security policy.
5. The system is portable and scalable because commercial untrusted systems are, in general, much more portable than trusted systems.

²Preliminary performance analysis indicates that SINTRA may outperform conventional database systems due to the parallel nature of the processing [McM94].

4 The SINTRA Models

In this section, we briefly discuss the security, transaction, and data models for SINTRA. These models are necessary in the development of any multilevel secure DBS and affect the decisions concerning the assignment of functions to each component of the SINTRA system.

4.1 Security Model

The security model used here is based on that of Bell and LaPadula [BeL76]. The model is stated in terms of subjects and objects. An object is a nonactive entity, such as a file, a relation, a tuple, or a field in a tuple. A subject is an active entity, such as a transaction or process, that can request access to objects. Every object is assigned a sensitivity classification, and every subject a clearance. Classifications and clearance are collectively referred to as security classes (or levels).

The database system consists of a finite set D of objects (data items) and a set T of subjects (transactions). There is a lattice S of security classes with ordering relation $<$. A class S_i dominates a class S_j if $S_i \geq S_j$. There is a labeling function L which maps objects and subjects to a security class:

$$L: D \cup T \rightarrow S$$

We consider two mandatory access control requirements:

(Simple Security Property) If

transaction T_i reads data item x then $L(T_i) \geq L(x)$.

(Restricted \star -Property) If transaction T_j writes data item x then $L(T_j) = L(x)$.

The simple security property allows a transaction to read data items if the security level of a transaction dominates the security level of data items. The restricted \star -property allows a transaction to write if the security level of a transaction is the same as that of data items (i.e., no write-ups or write-downs are permitted). Write-ups (i.e., T_i cannot write to data item x if $L(T_i) < L(x)$) are undesirable in database systems for integrity reasons (i.e., since a user cannot see what he has written, he may introduce an error)³.

4.2 Transaction Model

Traditionally, transactions are modeled as a sequence of read and write operations on data items. However,

³This is not to say that the frontend OS cannot write up to perform some functions of the overall DBS. This OS has its own compatible security model.

the traditional transaction model is not adequate to model transactions for the SINTRA system. Since the SINTRA treats each UBD as a black box, the global scheduler of the SINTRA system has very little knowledge about the behavior of the local scheduler (i.e., the scheduler of commercial DBS) or the physical layout of data. For example, the global scheduler has no knowledge about where a specific tuple is located or which physical page should be locked. Sometimes the tuples which will be modified are unknown until the computation based on existing data is completed.

We adopt a layered model of transactions, where a transaction is a sequence of queries, and each query can be considered as a sequence of reads and writes. For example, `replace` and `delete` queries can be viewed as a read operation followed by a write operation which must be executed atomically. `insert` can be viewed as a write operation, and `retrieve` can be viewed as a read operation. A layered view of two transactions T_1 and T_2 is shown in figure 2. Note this decomposition is similar to work of Weikum [Wei91] and Moss [Mos85].

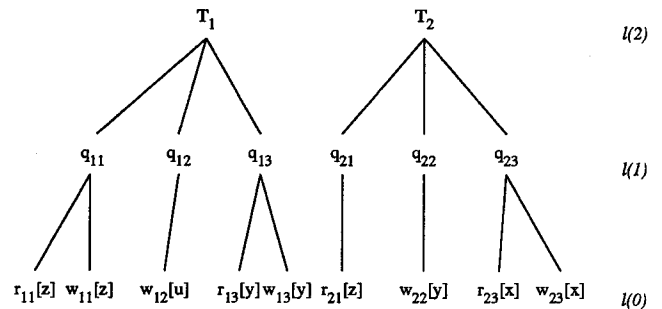


Figure 2: Layered model of two transactions.

Definition 1. A transaction T_i is a sequence of queries terminated by either a *commit*(c_i) or an *abort*(a_i), i.e., $T_i = \langle q_{i1}, q_{i2}, \dots, q_{in}, c_i \rangle$. Each query, q_{ij} , is an atomic operation and is one of `retrieve`, `insert`, `replace`, or `delete`.

To model the propagation of updates produced by a given transaction to higher security level databases, we define an *update projection*.

Definition 2. An *update projection* U_i , corresponding to a transaction T_i , is a sequence of update queries, e.g., $U_i = \langle q_{i2}, q_{i5}, \dots, q_{in}, c_i \rangle$ obtained from transaction T_i by simply removing all `retrieve` queries.

Note that no aborted transaction need be propagated. Hence, update projections are always terminated by a commit.

To describe concurrency control mechanisms, we adopt the following definition of *conflict*.

Definition 3. Two operations at the same layer **conflict** if they operate on the same data item and at least one of them is either **write**, **insert**, **delete**, or **replace**. Alternatively, two operations conflict if they operate on common data and not both are **retrieve** or **read** operations.

4.3 Data Model

Because the SINTRA prototype allows no modification of the commercial UBDs, it is necessary to treat the security labels of the data simply as additional data, conventionally stored. A more abstract data model might permit a more refined representation of the semantics, but the SINTRA prototype data model is limited to representing labels as values of label attributes.

Typically, labels can be associated with either values of the individual ordinary attributes or with an entire tuple. The SINTRA data model is based on an element-level classification scheme. A multilevel relation scheme is denoted by

$$R(A_1, C_1, A_2, C_2, \dots, A_n, C_n, TL)$$

where each A_i is a data attribute over domain D_i , each C_i is a classification attribute for A_i and TL is the tuple-level attribute. The domain of C_i and TL is specified by a range $[L_i, H_i]$ in the security lattice. $R(A_1, A_2, \dots, A_n, TL)$ is the underlying relation as viewed by the user.

Let $t[A_i]$ denote the value corresponding to the attribute A_i in tuple t , and similarly for $t[C_i]$ and $t[TL]$. $t[TL]$ in the SINTRA system simply specifies that a tuple t is generated or modified by a $t[TL]$ -level user. On the other hand, $t[C_i]$ specifies that the corresponding $t[A_i]$ originated from $t[C_i]$ -level. Operations on the database that allow retrievals and updates are presented in [KCF93a]. There are, in addition, constraints among the values of the C_i and TL [KCF93a] which are of little importance for explaining the prototype.

5 The SINTRA Scheduler

The replicated architecture provides mandatory access control by physically separating data. However,

this approach introduces another problem, namely the maintenance of mutual consistency of the replicas. Our research suggests that accepting the replica consistency problem in return for a virtually free high assurance, strong protection mechanism is a choice well-made. In this section, we introduce the structure and functions of the global scheduler. True distribution of these functions is, in the abstract, of little consequence, but has been done in the prototype in a particular way to maximize performance.

5.1 Responsibilities

Since each UBD in a replicated architecture contains data from lower levels, update queries have to be propagated to higher-security-level databases to maintain the consistency and currency of the replicated data. If this propagation of update queries is not carefully controlled, inconsistent database states can be created among backend databases. Consider that two lower level update transactions T_i and T_j are scheduled with serialization order $\langle T_i, T_j \rangle$ at the lower level backend database system. Since these two transactions are update transactions, they have to be propagated to the next higher level. If these two transactions are scheduled with serialization order $\langle T_j, T_i \rangle$ at the next higher level, an inconsistent database state between these two backend databases may be created by the execution of conflicting operations at the higher level. It can be demonstrated that even the serialization order of non-conflicting transactions has to be maintained to preserve one-copy serializability [KFC92]. This is a parallel result that has been reported in the context of multidatabase systems [Du93]. Therefore, the serialization order introduced by the local scheduler at the user's session level must be maintained at the higher level UBDs.

The concurrency control algorithm which the SINTRA prototype uses has two types of schedulers, global and local schedulers. The global scheduler enforces data consistency among different security levels. On the other hand, the local scheduler, which is the unmodified commercial concurrency controller of a UBD, manages transactions and update projections at that UBD. The local scheduler deals with layer $l(0)$ in figure 2, and the global scheduler deals with layer $l(1)$ and upper layers. The global scheduler detects conflicts at level $l(1)$. Therefore, no knowledge of the specific items to be accessed or even the granularity of the lower-level concurrency controller is needed or used by the global scheduler.

SINTRA's global scheduler resolves the data-

inconsistency problem by guaranteeing that the serialization order introduced by the local scheduler at the user's session level is maintained at the higher level UBDs. In summary, the global scheduler performs the following tasks:

- Receive queries from the query preprocessor and the global scheduler of lower security levels, and send them to the backend database.
- Guarantee that the serialization order introduced by the local scheduler at the user's session level is maintained at the higher level UBDs.
- When a transaction is committed, send an update projection to higher security level backends.

Since user transactions and update projections are submitted independently, their serialization orders are not known to the global scheduler. Hence, the *take-a-ticket*[Geo91] operation is used to find the serialization order among update projections and user transactions. Generalized algorithms and a theory of a global scheduler for the SINTRA database system have been presented by Kang, Froscher, and Costich in [KFC92].

As previously mentioned, the global scheduler resides partially in the TFE and partially in the UBD. This was done for performance rather than theoretical reasons.

5.2 Structure

The methodology used for developing the SINTRA global scheduler closely resembles the object-oriented development method [Boo86]. We identify many objects, queries, transactions, processes, etc., and establish the relationships among these objects. Many layers are also introduced to hide lower-level details. C++ has been chosen as our main implementation language because it provides the capabilities of data hiding and abstraction of interface. Scheduler components which are executed on the frontend are written in C because the XTS-200 provides neither a C++ interpreter nor the C compiler which can compile C code that is generated by a C++ interpreter.

The process architecture of the global scheduler is as follows (see also figure 3):

- 1. Terminal:** User terminals or workstations.
- 2. Frontend Connector:** Establish a virtual connection between the user and the backend depending on the user's session level.

- 3. Database Server:** When user login is requested, spawn a database server child to service the request.
- 3a. Database Server Child:** Ensure connections among user, preprocessor, and user transaction scheduler.
- 4. Preprocessor:** Query modification (see section 7).
- 5. User Transaction Scheduler:** Submit user transactions to the ORACLE database. Send the response from ORACLE to the user. Also send an update projection to the propagation scheduler if a user transaction is committed.
- 6. Propagation Scheduler:** Receive transactions and send them to the corresponding frontend update projection receiver according to the serialization order.
- 7. Projection Receiver:** Receive update projections from the propagation scheduler and store them for retrieval by the projection sender.
- 8. Projection Sender:** Read-down to get the update projections which are in the lower level projection receiver and send the projections to the update projection scheduler.
- 9. Update Projection Scheduler:** Receive update projections from the lower level backend, submit them to the ORACLE database, and send them to the propagation scheduler.

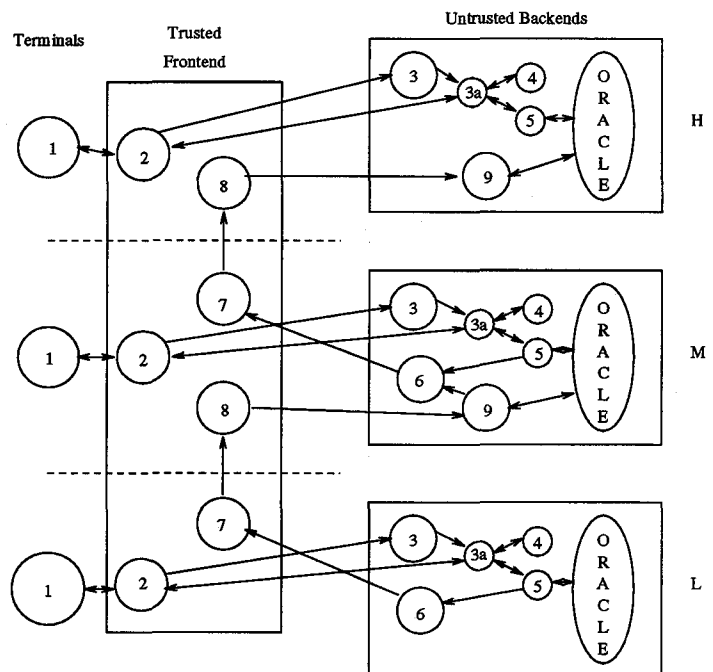


Figure 3: The Process-level Architecture.

Our process architecture has been implemented using C++ objects, and complex interprocess communication details have been hidden by an interprocess communication (*ipc*) class. A detailed description of each object and methods is given by Kang and Peyton in [KaP93a].

6 Recovery Mechanisms

Recovery in the SINTRA MLS DBS is largely dependent on the recovery mechanisms of the commercial DBSs that are used at the backends. Without the propagation of update projections, SINTRA appears to be a single-level DBS. Hence there is no need for recovery mechanisms between users and the UBD. However, the SINTRA database system needs recovery mechanisms that guarantee the delivery of update projections from user transaction scheduler to the next level DBS (i.e., path 5 → 6 → 7 → 8 → 9 → DBS in figure 3).

To ensure that committed transactions will not be lost (except by media failure), processes responsible for the propagation of update projections use persistent queues to log update projections. Update projections are kept in the log until the next process within the propagation chain acknowledges that it has received the update projection. An acknowledgment from a receiver guarantees the sender that the receiver has safely processed the update projection.

The above technique works without creating a covert channel if the sender and receiver are at the same security level. However, if the same technique is used when the security level of the receiver is higher than that of the sender, then there is a covert channel. Alternatively, when a message is delivered to a receiver, it sends either an *ack* or control back to the sender acknowledging that the message is in stable log. However, if the security level of the sender is lower than that of the receiver, then the timing of an *ack* can be used as a covert timing channel. A detailed description of the problem and proposed solutions are presented by Kang and Moskowitz [KaI93].

When the system recovers from a failure, it needs to know the status of DBS (i.e., the last transaction that has been committed) so that the system can guarantee the consistent states among DBS and persistent queues in the global scheduler. The SINTRA recovery mechanism uses the *ticket* which was introduced in section 5.1 to examine the status of the DBSs. When the system recovers, the processes of the global scheduler examine the status of the DBS and their persistent

queues (i.e., sort out which transactions are committed and which ones are not). Then the update projection scheduler will submit update projections which have not been committed to the DBS and the propagation scheduler will send committed transactions to the projection receiver. A detailed design description and high-level pseudo-code are presented in [KaP93c].

7 The Query Preprocessor

In this section, we explain the need to modify user queries and the internal structure of the query preprocessor. The SINTRA query preprocessor is written in C++. YACC++ and LEX++ are also used to build the query parser.

7.1 Responsibilities

The SINTRA query preprocessor plays an important role in maintaining data consistency among different backend databases, preserving data integrity, and bridging the semantic gap between conventional and multilevel-secure databases. The SINTRA query preprocessor has the following responsibilities:

1. In the SINTRA database system, if a high-level user is allowed to modify low data which are located at the high-level backend database, then inconsistent database states between high and low backend databases can be created. Therefore, the query preprocessor must inspect, and either reject or modify users' update queries so that the backend database system only modifies users' login level data — it is also assumed that no *write-up* is allowed.
2. There is also some information which can be modified only by the system although it can be disclosed to the user. For instance, information about the classification of a tuple (TL attribute values) cannot be modified by the user. It is the responsibility of the query preprocessor to guarantee the integrity of such data.
3. SINTRA uses conventional relational database systems as backend databases. These conventional relational databases use SQL, which is based on the conventional (single-level) relational algebra and the semantics of conventional update operations [Ull82]. On the other hand, a multilevel relational database is based on a multilevel relational algebra and the semantics of multilevel relational update operations. Therefore, a

SINTRA user query which is posed to an MLS database in multilevel SQL must be translated into other queries (based on the conventional SQL used by the UBD) that conform to the semantics of conventional update operations.

To perform the above responsibilities, the SINTRA query preprocessor intercepts, inspects and modifies user queries before they are submitted to the ORACLE database system.

7.2 Structure

Consider the following user query, `delete from R where A1 = 5 and TL = 'L'`. It is a legitimate query if an *L*-user (the user whose session level is *L*) issued the query. However, if an *H*-user issues the same query then the query must be blocked (i.e., otherwise this query creates an inconsistent copy of *L*-data at the *H*-backend). Therefore, when the SINTRA preprocessor makes decisions on how a user query should be modified and what kind of query has to be blocked, it has to be based not only on the syntax of the individual query but also on the session level of the user who issues the query. Hence, simple syntactic checking is not sufficient to determine if a query is legitimate or not. We chose the following process organization:

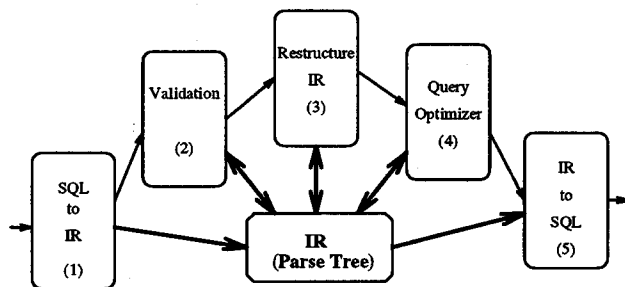


Figure 4: The Structure of the Query Preprocessor.

First, a user query is parsed and converted into an internal representation (IR) which is a parse tree. All syntactically invalid queries and some invalid queries which can be detected without knowing the security level of the user will be rejected at this stage. For example, a user query `update R set a = 5, a# = 'H' . . .`⁴ will be rejected at this stage, because the query

⁴The SINTRA preprocessor needs to distinguish regular attributes from classification attributes. Hence, the SINTRA preprocessor reserves one character (e.g., '#') for preprocessor use only. For example, a user defines an attributes *a* in a specific relation, then its corresponding classification attribute will be *a#*.

preprocessor knows that *a#* is a classification attribute, and users are not allowed to modify classification attributes.

The second process, validation, of figure 4 inspects all syntactically valid queries again to check if they can create any inconsistent replicas. For example, `delete from R where a = 5 and T#L = 'L'` by a *H*-user will be rejected at this stage because the *H*-user tries to delete *L*-data. Hence, responsibilities (1) and (2) in section 7.1 are accomplished by the first and second stages.

The third process, restructure IR, performs the main mission of the SINTRA query preprocessor, i.e., query modification. It will modify parse trees based on the SINTRA multilevel relational algebra and the semantics of update operations for multilevel relations [KCF93a]. Consequently, functions (3), (4), and (5) in section 7.1 are accomplished at this stage. For example, a *H*-user's query, `delete from R where b = 'xxx'`, will be translated into `delete from R where b = 'xxx' and TL = 'H'`. A detailed design description of the SINTRA query preprocessor appears in [KaP93b].

The fourth process, query optimizer, optimizes parse trees based on the knowledge of the implementation. Finally the fifth process, IR to SQL, converts parse trees into conventional SQL before the query is submitted to ORACLE at the UBD.

8 The User Interface

The SINTRA system uses a client-server architecture for its user interface. The user interface resides on the client side and the DBS resides on the server side. Network interface units reside in both clients and servers (see figures 1 and 5).

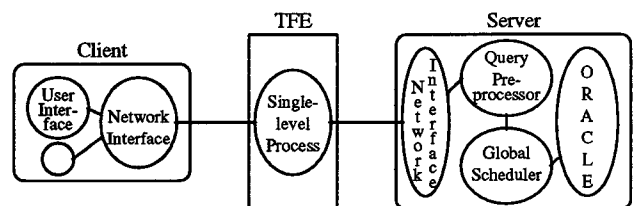


Figure 5: A Detailed Description of Network Interface.

The network interface process is responsible for directing queries from the user interface to the proper server, and directing responses from the server to the proper client. The single-level process in TFE, which resides

between clients and servers, assures that the communication between two network interfaces does not violate the SINTRA security policy.

Note that each client is dedicated to one security level, and identification and authentication procedures are still performed by the TFE with the current configuration. However, this restriction (i.e., a client can login only one security level) can be removed.

9 Status and Conclusions

The prototype has demonstrated the feasibility of replication as a path to high-assurance security, good performance, reduced maintenance cost, and portability. This architecture uses physical separation as a protection measure. Alternatively, the mandatory access control of the SINTRA approach depends on the MAC of the TFE and the physical separation of UBDs, and the discretionary access control (DAC) of the SINTRA approach depends on the DAC of UBD.

Since a system based on this architecture can be built from commercial DBSs, it is very portable and maintenance is relatively simple. High performance is achieved by storing all information that a user can legitimately view at one backend.

9.1 Status of Prototype

The current status of the SINTRA implementation is as follows:

- The design and implementation of the global scheduler are both finished.
- The design and implementation of the query preprocessor are both finished (i.e., a subset of SQL and full operations that are discussed in [KCF93a] have been implemented).
- The design of the user interface is finished. There are many commercial user interface tools for the ORACLE database. We hope to use those tools for the SINTRA. However, almost all user interface tools for ORACLE use *SQL*Net* for communication between the interface tool itself and ORACLE. Since the SINTRA preprocessor needs to intercept, inspect and modify user queries before these are submitted to ORACLE⁵, the SINTRA preprocessor needs to know the internal format of

⁵The query decomposer of the KSR1 computer performs the same task to parallelize the queries.

packets that *SQL*Net* uses. Until we have that information, we will proceed to implement our own user interface.

- The design of the recovery mechanism is finished and the implementation of the mechanism is in progress.

9.2 Lessons Learned

This prototyping exercise has demonstrated the feasibility of the replicated architecture approach first described in [Air83]. Prior to the SINTRA project, none of the high assurance approaches described in [Air83] had been demonstrated. We have learned several important lessons from our development of the SINTRA prototype.

- The strong separation of concerns inherent in the replicated architecture allows the use of commercial DBSs without modification. This means that the replicated approach can accommodate new database technology without significant porting effort or reengineering.
- Several replica and concurrency control algorithms have been developed for the project and have been proven correct. The implementation of the SINTRA global scheduler moved our results from theory to practice and showed that the replicas remained consistent when the database was updated.
- The trusted OS for the TFE required no modification. Little additional software was developed to run on the TFE. We were successful in minimizing our dependence on the TFE.
- The replicated architecture approach imposes fewer data model restrictions than kernelized approaches.
- Preliminary performance analysis and simulation results indicate that the replicated architecture provides good performance.
- This approach does require more hardware than the kernelized approach. As hardware costs become lower, it will be a trade-off decision for users to decide whether improved performance and usability are worth the additional hardware cost.

We hope that the SINTRA prototype will serve as the catalyst for future high-assurance multilevel

database research. Our plan is to use B1 DBSs running on compartmented mode workstations as backend databases. This configuration will provide the B1 separation among compartments, and the B3 separation among security hierarchies.

References

- [Air83] Multilevel Data Management Security, Air Force Studies Board, Commission on Engineering and Technical Systems, National Research Council, National Academy Press, Washington, D.C. (1983).
- [BeL76] Bell, D. E., and LaPadula, L. J. Secure computer systems: Unified exposition and *multics* interpretation. The Mitre Corp, (1976).
- [Boo86] Booch, G. Object-Oriented development. IEEE Transactions on Software Engineering, 12, 2 (1986).
- [DoD85] Department of Defense National Computer Security Center, *Trusted computer system evaluation criteria*, DoD5200.28-STD (1985).
- [Du93] Du, W., Elmagamid, A. K., Kim, W., and Bukhres, O. Supporting consistent updates in replicated multidatabase systems. The VLDB Journal, 2, 2 (1993).
- [Den85] Denning, D. Commutative filters for reducing inference threats in multilevel database systems. The IEEE symposium on Security and Privacy (1985).
- [Geo91] Georgakopoulos, D., et al. On serializability of multidatabase transactions through forced local conflicts. Conference on Data Engineering (1991).
- [Jen89] Jensen, C., et al. SDDM: A prototype of a distributed architecture for database security. Conference on Data Engineering (1989).
- [KFC92] Kang, M. H., Froscher, J. N., and Costich, O. A practical transaction model and untrusted transaction manager for multilevel-secure database systems. The Eighth IFIP Workshop on Database Security (1992).
- [KCF93a] Kang, M. H., Costich, O., and Froscher, J. N. The replicated architecture data model: structure and operation. Internal Report (1993).
- [KCF93b] Kang, M. H., Costich, O., and Froscher, J. N. Using object modeling techniques to design MLS data models. Security in Object-Oriented Systems, Springer-Verlag, ISBN 3540198776 (1994).
- [KaI93] Kang, M. H., and Ira S. Moskowitz. A pump for rapid, reliable, secure communication. ACM Conference on Computer and Communications Security (1993).
- [KaP93a] Kang, M. H., and Peyton, R. Design documentation for the SINTRA global scheduler. Naval Research Laboratory Memo Report 5542-93-7362 (1993).
- [KaP93b] Kang, M. H., and Peyton, R. Design documentation for the SINTRA preprocessor. Internal Report (1993).
- [KaP93c] Kang, M. H., and Peyton, R. Design documentation for the SINTRA recovery mechanism. Internal Report (1993).
- [Lun90] Lunt, T., et al. The SeaView security model. IEEE Transactions on Software Engineering, 16, 6 (1990).
- [McM94] McDermott, J. and Mukkamala, R. Performance analysis of transaction management algorithms for the SINTRA replicated-architecture database system. In Database Security, VII: Status and Prospects, eds. Thomas Keefe and Carl Landwehr, North-Holland (1994).
- [Mos85] Moss, E. Nested transactions, an approach to reliable distributed computing. The MIT Press (1985).
- [OcG89] O'Connor, J., and Gray, J. A distributed architecture for multilevel database security. National Computer Security Conference (1988).
- [Wei91] Weikum, G. Principles and realization strategies of multilevel transaction management. ACM Transactions on Database Systems, 16, 1 (1991).

The SeaView Prototype: Project Summary

Teresa F. Lunt and Peter K. Boucher*
Computer Science Laboratory
SRI International, Menlo Park, California 94025

Abstract

The SeaView multilevel-secure database management system prototype was completed by SRI during a two-year project that followed the original three-year SeaView project to design a multilevel database system meeting the requirements for Class A1 of the U.S. DoD *Trusted Computer System Evaluation Criteria*. The prototype described here utilizes existing database technology as well as a commercially available general-purpose trusted operating system. On this base we constructed a layer of software to implement multilevel relations and to support user interaction through the MSQL language, an extension of the standard Structured Query Language (SQL). The prototype was successfully demonstrated at Rome Laboratory and at NSA.

1 Introduction

The SeaView prototype multilevel-secure relational database system [1] is based on the SeaView formal model and preliminary design [2, 3, 4, 5, 6]. The prototype is intended to serve as a feasibility demonstration whose purpose is to validate the SeaView theoretical model and system requirements and to demonstrate that the prototype is suitable for engineering development. This work was sponsored by the U.S. Air Force, Rome Laboratory.

The SeaView prototype can handle data with a variety of classifications and serve users with a variety of clearances. Users at different access classes see different views of the same multilevel table. The prototype provides element-level labeling; individual data elements in a multilevel relation can be labeled with an access class. SeaView extends the traditional relational data model to explicitly include the classifications of individual data elements and the security rules of access to stored and derived data [4]. Each multilevel relation has a primary key defined for it, which is an attribute or uniformly classified group of attributes. SeaView's interpretation of database integrity in the context of multilevel security allows database integrity to be achieved without introducing covert inference channels. We introduced the notion of polyinstantiation, which prevents low users from inferring the existence of high data objects. The prototype provides a standard implementation-independent multilevel query language called MSQL (Multilevel Structured Query Language) for defining and manipulating multilevel relations. We have defined the MSQL syntax to be an extension of the standard SQL language [1].

*The work reported here was funded by Rome Laboratory under U.S. Government contract number F30602-89-C-0158.

The prototype was designed for Class A1 assurance [7]; it has been implemented on a Class B1 operating system platform, but is capable of evolution to Class A1 when hosted on a Class A1 platform. SeaView's design takes the most secure approach possible, in that all database processing is carried out by single-level subjects. Multilevel relations are implemented as relational views over single-level base tables that are transparent to users¹. Trusted ORACLE is used to manage the single-level relations. (Trusted ORACLE was selected because it is the only available multilevel database management system which includes a mode of operation for which no trust is required in the database system itself; see Section 3.2.) An underlying security kernel enforces the mandatory security rules. We use the Sun Compartmented Mode Workstation (CMW), and store the decomposed single-level relations in CMW files of the corresponding access class, managed by the CMW Trusted Computing Base (TCB). The access class of any particular data element in a multilevel relation is derived from the access class of the single-level relation in which the data element is stored; this in turn matches the access class of the file in which it is stored, which is known to the CMW. Thus, individually labeled data elements need not be stored in individually labeled storage objects, as was assumed prior to SeaView.

Our approach allows multilevel `select`, `update`, `insert`, and `delete` operations to be decomposed into corresponding single-level operations on the single-level base relations, and lends itself to a design that uses a commercially available relational database management system (RDBMS) for the single-level relations. The decomposition is transparent to the user, who considers the multilevel relations to be stored relations. Thus, the SeaView model extends the application-independent integrity rules of the relational model — namely, entity integrity and referential integrity — to multilevel relations; it allows application-dependent integrity rules to be defined on multilevel relations; and it ensures that updates of multilevel relations are well defined. In addition, the SeaView model constrains multilevel relations by a third application-independent integrity rule, polyinstantiation integrity, which specifies consistency for polyinstantiated tuples and elements [4]. We believe that our design includes the basic functionality required by multilevel applications utilizing an integrated collection of data classified at different access classes.

The advantages of the SeaView approach over other trusted database system architectures are:

- *Element-level labeling.* Element-level labeling allows a more natural modeling of multilevel data for many applications, where it is desirable to represent multilevel entities in the database. For example, multilevel aircraft could be represented, where the existence of the aircraft, and their names, identifiers, and ranges are SECRET, but their on-board weapons and their cargo are TOP-SECRET. Element-level labeling allows the programmer to think about such multilevel entities in a more natural way, and it relieves the programmer of the burden of mapping each multilevel entity type to tuple- or table-labeled objects in the application code. It also relieves the programmer of implementing in the application code, for each multilevel entity type, the rematerialization of multilevel entities from tuple- or table-labeled objects before presenting the multilevel data to the user. It thereby reduces complexity, and thus cost and chances for errors, in the application code.
- *Evolvability to high assurance.* SeaView's use of the strict trusted subsets architecture allows evolution to higher assurance classes when SeaView is ported to a high-assurance platform

¹That is, users see only the multilevel relations and are not aware of and cannot directly access the single-level base tables that implement them.

(see Section 3.2). Several high-assurance platforms are now becoming available, and there is a clear need for high-assurance systems even if only a few hierarchical classifications must be handled by the system [8].

- *Ease of evaluation/accreditation.* The SeaView software enforces neither mandatory nor discretionary access control and is not trusted for either mandatory or discretionary security. Mandatory security is enforced entirely by the underlying trusted operating system, and discretionary security is enforced entirely by Trusted ORACLE. Other security-relevant functions, such as authentication and audit, are also performed entirely by those underlying products. Thus, SeaView can be added to an existing system composed of a trusted operating system running Trusted ORACLE without affecting its security rating.

We originally targeted our prototype to run on GEMSOS, a Class A1-evaluatable platform. We produced a design and nearly completed the implementation of the SeaView layer for the prototype design that included GEMSOS. Under separate contract to NSA, Oracle Corporation and Gemini Computers were to have ported Trusted ORACLE to GEMSOS. However, that project has not yet been completed, so that we had to find an alternative platform. After thorough investigation, we decided to retarget the existing SeaView prototype to the Sun CMW. Our criteria for selecting a new platform were (1) Trusted ORACLE must be available on it; (2) it must allow untrusted subjects to query labels, perform label comparisons, and perform other related label operations; and (3) it should be available at low cost. We selected the Sun CMW because at the time it was the only platform that satisfied these criteria. This retargeting was accomplished in about three months. The SeaView design and architecture allow the evolution to Class A1 if the CMW is replaced with an A1 operating system.

The prototype is written entirely in the C programming language. The only platform- and product-dependent software are six Trusted ORACLE Pro*C modules and one Sun CMW label module. This platform- and product-independent implementation approach will allow SeaView to be readily retargeted and ported to other secure platforms and trusted database engines.

Two other efforts have been made to build Class A1 relational database systems. A group at Secure Computing Corporation (SCC) is implementing LOCK DBMS, which is a port of Trusted ORACLE to LOCK/SNS that uses LOCK's type enforcement mechanism [9]. ASD is a prototype developed at TRW. It provides row-level labeling and is designed to run on the Army Secure Operating System (ASOS). The database system TCB runs as a trusted subject on the underlying trusted operating system and enforces both mandatory and discretionary security [10]. In addition, several vendors have announced or released products designed to meet some of the U.S. DoD criteria.

2 Functionality of the SeaView Prototype

The SeaView prototype provides an interactive MSQL interface that allows the user to enter MSQL queries from the keyboard, as well as a forms interface allowing the user to enter MSQL queries by using the mouse and filling in a form selected from a menu. The prototype also includes an MSQL data dictionary containing definitions of user-defined multilevel relations as well as non-user-visible information on the mappings of those multilevel relations onto the underlying single-level Trusted ORACLE tables and views. The MSQL query processing functions supported are the `select`, `update`, `insert`, and `delete` statements on multilevel relations. The prototype allows a user to create multilevel relations using the `create table`

statement. Execution of the `create table` statement automatically creates the requisite set of underlying single-level Trusted ORACLE tables and views and makes the necessary entries in the MSQL data dictionary. The SeaView prototype enforces multilevel entity integrity and multilevel referential integrity, as defined by the SeaView model [4].

Features supported by the prototype's MSQL include the following:

- *Label Display.* Users can display element and tuple labels in a query result by using the keyword `showlabel` in the query. The function `class()` returns the access class of the attribute specified as its argument. The predefined column `rowlabel` returns the tuple class for each returned tuple.

- *Access class operators.* The prototype provides the comparator operators `dom (>=)` (dominates), `str_dom (>)` (strictly dominates), `dom.by (<=)` (dominated by), `str_dom.by (<)` (strictly dominated by), and `uncomp (<>)` (noncomparable to).

- *Access class functions.* The prototype provides the functions `least_UB()` (least upper bound) and `greatest_LB()` (greatest lower bound). These are used analogously to the SQL `max` and `min` functions. A single attribute name is given as the argument to these functions.

- *Partial class functions.* These functions extract the integrity and secrecy components of the access class as well as the level and set of categories of an integrity or secrecy component of an access class. They are `sec()` (secrecy component), `intg()` (integrity component), `sec_lvl()` (secrecy level), `intg_lvl()` (integrity level), `sec_cat()` (set of secrecy categories), and `intg_cat()` (set of integrity categories). Functions such as greater than, less than, equality, and set membership are also provided for the values returned by these functions. A single attribute name is given as the argument to these functions.²

- *Class aggregate functions.* The prototype provides the functions `highest()`, `lowest()`, `LUB()` (least upper bound), and `GLB()` (greatest lower bound) that can be applied to a list of supplied values or a set of values returned by an embedded subquery.

3 Project Goals and Achievements

The SeaView prototype has successfully demonstrated a proof-of-concept for element-level labeling, the TCB subsets architecture, polyinstantiation, multilevel relation decomposition and materialization, the MSQL language, and high assurance.

3.1 Element-Level Labeling

Element-level labeling allows the independent classification of individual atomic facts, and also allows real-world multilevel entities to be directly represented in SeaView's multilevel relations. The SeaView prototype demonstrates:

- *Element-level labeling.* Individual data elements (the intersection of a row and column in a multilevel table) are labeled with a security classification. In addition, each column (or group of columns) of a table can be assigned a range of security classifications within which all labels for data in that column must fall. Tuple labels display the overall security classification of each tuple (row) in a table or query result. These security labels include a secrecy component and an integrity component. Each label component can contain a hierarchical level and a

²These partial class functions are implemented in the SeaView prototype but do not work because of limitations of the Trusted ORACLE beta version that we used; upgrading to a production version of Trusted ORACLE should make these functions available.

set of categories. Labels can be displayed or suppressed for any query, at the user's option. Moreover, individual column labels and/or tuple labels can be displayed or suppressed within a query result, at the user's option. The labels are advisory; element labels indicate the security level of the underlying storage object (file) in which the associated data element is stored, and tuple labels indicate the least upper bound of data used to form or compute the tuple, but the overall classification of the set of data returned from any query is indicated by the sensitivity label in the CMW window banner. This is appropriate since the query itself, which is typed in by the user, must be treated as classified at the subject access class, which thus provides a lower bound for the overall security level of the query result (since the text of the query is used as input in computing the query result).

- *Selection of data based on their labels.* SeaView provides the ability to select data for retrieval based on values of the element and tuple labels associated with the data. The MSQL query language includes an access class data type and provides operations for that type. For example, data can be selected based on their access classes, their secrecy components, their integrity components, their secrecy categories, or their integrity categories. Operations are provided to select data based on label comparisons (e.g., select the names of all employees for which the secrecy class of the salary dominates the secrecy class of the employee number) or other label computations.³

- *Display of data appropriate for the subject class.* SeaView displays only data whose element and tuple labels are dominated by the access class of the subject that issued the query.⁴ Moreover, in computing a query result to be returned, SeaView uses only data whose classification⁵ is dominated by the subject class.

- *Assignment of the subject class to newly entered data.* SeaView assigns the subject class to element labels for updated and inserted data.

- *Prohibition of modification of data labeled low by high subjects.* SeaView allows a subject to update only those data elements whose label equals the subject class⁶. A subject can delete a tuple only if the access class of the primary key elements equals the subject class.⁷

- *Reasonable storage requirements.* Prior to the original SeaView project, element-level labeling was not thought to be feasible, because it was considered necessary to store a security label with each data element, thereby at least doubling the amount of storage necessary for the database. SeaView does not store an element label with each data element. Instead, it collects large groups of data elements with the same security level into larger storage objects (files), and obtains the label for the element from the operating system's label for the storage object. This means that a single stored label can be common to a very large number of data elements, so that very little additional space is required for labels.

³Because of limitations of the particular software products employed in the SeaView prototype, which are *beta* or early developers' versions of the Sun CMW trusted operating system and of the Trusted ORACLE trusted database management system, the operations involving access class components cannot be used, although they are implemented in the SeaView code and should work with later production versions of these same products.

⁴In the CMW, the access class of the subject that issues a query is the same as the sensitivity label for the window in which the query was typed.

⁵The classification of data, for purposes of access control, is indicated by the access class of the storage object (file) in which it is contained.

⁶Updating a low data element results in polyinstantiation of the element - that is, a high version of the element is created, and the low version remains unchanged.

⁷SeaView also requires the access class of the primary key elements to be dominated by the access classes of all other data elements in the tuple; this guarantees that high subjects cannot delete low data.

- *Element-level labeling without the need to trust the entire system.* Prior to the original SeaView project, it was thought that to provide element-level labels, the entire database system would have to be trusted. SeaView's approach uses strict TCB subsets (see Section 3.2, below) and provides advisory labels; the approach requires only the operating system TCB to be trusted for mandatory security.

3.2 High Assurance and the TCB Subsets Architecture

The "trust" in trusted computer systems rests on the ability to provide convincing arguments or proofs that the security mechanisms work as advertised and cannot be disabled or subverted. In building multilevel database systems, providing such assurance is especially challenging because large, complex mechanisms may be involved in policy enforcement. To satisfy mandatory security requirements, we assign access classes to processes, or subjects, derived from the clearance of the user on whose behalf the subject is operating. Traditional practice is to segregate the security-relevant functions into a security kernel or reference monitor. The reference monitor mediates each reference to an object by any subject, and allows or denies the access according to a comparison of the access classes associated with the subject and with the object. The reference monitor must be tamperproof, it must be invoked for every reference, and it must be small enough to be verified to be correct and secure with respect to the policy it enforces. A high degree of assurance must be provided that the mandatory security mechanisms not only control access to sensitive information, but also that they enforce confinement, or secure information flow. The reference monitor forms the core of the TCB, which contains all security-critical code. The U.S. DoD *Trusted Computing System Evaluation Criteria* includes requirements for "minimizing the complexity of the TCB," and "excluding from the TCB modules that are not protection-critical," so that the reference monitor is "small enough to be verifiable" [7]. Without such a requirement, the high degree of assurance required would not be feasible.

So as to produce a design that allows the prototype to evolve to Class A1 assurance, we have adopted a design approach that reuses and builds on previously built and verified trusted systems. SeaView builds a database system on top of a reference monitor for mandatory security, so that the mechanisms responsible for enforcing multilevel security are segregated in the reference monitor, which is small enough to be verified. The approach has been called *strict TCB subsetting* [11, 12] and has sometimes been called *constrained* or *self-contained* TCB subsetting. The TCB subsetting concept evolved from earlier work on extensible TCBs [13].

The TCB subsetting approach reuses and extends previously built and verified trusted systems. It is motivated by the need to be able to extend a TCB by building on an existing one without disturbing its basis for evaluation. This is essential when a vendor wants to build a trusted database system on another vendor's trusted operating system.

SeaView demonstrates the feasibility of that approach to building trusted systems. The strict TCB subsetting approach structures the TCB in nonbypassable layers, with each layer enforcing its own policies and with each layer constrained by the policies enforced by the layers beneath it. In particular, the lowest layer is a mandatory TCB that enforces mandatory security for all the layers above it and which contains all code trusted with respect to mandatory security. With the exception of the lowest TCB layer, all TCB layers are untrusted with respect to mandatory security. The use of the strict TCB subsets approach allows the reuse of existing database and OS/TCB technology; it also can lead to a much faster product evaluation, since the evaluation process can take advantage of the known, evaluated properties of the reused technology.

The difficulty of using the strict TCB subsetting approach is great for those database system vendors whose processing model consists of a single server process servicing all user requests. A more natural model for a self-contained TCB subsetting approach is a processing model consisting of multiple database server instances, each servicing the requests of subjects at a single access class. Trusted ORACLE uses a multi-instance processing model that allows different database instances to have different security levels, so that the multilevel database system can operate as a collection of single-level processes.

Although it is not strictly necessary for Class A1 assurance with our strict TCB subsets approach, we developed a formal specification for SeaView [14] and performed a partial formal verification of the formal specification to the SeaView model properties [3, 15] using the EHDM verification tools [16]. We also produced a Formal Top-Level Specification (FTLS) to code correspondence report [17].

The SeaView prototype demonstrates the following properties:

- *Strict TCB subsets architecture.* The SeaView prototype uses the strict TCB subsetting approach. The lowest TCB layer consists of the Sun CMW operating system's TCB; this lowest system layer is responsible for enforcing the mandatory access control policy.⁸ The next-lowest TCB layer is the Trusted ORACLE TCB.⁹ This TCB layer is responsible for enforcing the discretionary access control policy and is completely constrained by the underlying CMW mandatory TCB; that is, it is completely untrusted with respect to mandatory security. The highest system layer, consisting of the SeaView software, is responsible for enforcing multilevel relational integrity properties. This layer is completely constrained by the Trusted ORACLE discretionary TCB and the CMW mandatory TCB; that is, it is completely untrusted with respect to both mandatory and discretionary security.

We reemphasize the following consequences of the approach. Mandatory security is enforced only by the TCB of the underlying operating system. There is no trust for mandatory security in the database system software; that is, the database system software operates entirely as single-level untrusted subjects. Separate single-level database processes are required for each active access class (i.e., for each access class at which a user is currently logged in). Global integrity constraints that span access classes cannot be enforced, because this would either result in signaling channels or require trusted subjects [18, 2, 3]. In particular, polyinstantiation (see Section 3.3) cannot be avoided because to do so would require a signal to be sent to a low user based on the presence or absence of high data. New data are assigned the subject class (because the database software is untrusted with respect to mandatory security, and no writing down is permitted). Element and tuple labels are advisory (since there is no trust for mandatory security in the database software).

- *Architecture evolvable to Class A1.* The strict TCB subsets approach allows SeaView to straightforwardly evolve to a higher-assurance system. Because all mandatory security enforcement is performed in the underlying operating system TCB, replacing the CMW with a

⁸SeaView's design does not require the use of the CMW; any trusted operating system that can support Trusted ORACLE in OS MAC mode can be substituted. The use of the strict TCB subsets approach means that the substitution of a higher-assurance trusted operating system will result in an equally high assurance SeaView system.

⁹The Trusted ORACLE trusted database management system product can operate in two modes: *OS-MAC* and *DBMS-MAC*. Trusted ORACLE's OS-MAC mode relies entirely on the mandatory TCB of the underlying operating system for the enforcement of mandatory access control; in this mode, Trusted ORACLE processes run as untrusted software, with respect to mandatory security. In DBMS-MAC mode, on the other hand, Trusted ORACLE assumes some responsibility for the enforcement of mandatory security. In SeaView, Trusted ORACLE operates in OS-MAC mode.

higher-assurance platform will automatically give a concomitant degree of assurance for mandatory security for the resultant SeaView system. Thus, no research or technology development is needed for SeaView to evolve to a Class A1 system. Once suitable high-assurance platforms become available and Trusted ORACLE is ported to those platforms, SeaView will be readily available on those platforms, providing high assurance for mandatory security.

- *Lowest possible security risk.* The use of strict TCB subsetting provides the greatest degree of assurance possible for mandatory security. This approach allows us to use the operating system's TCB to enforce mandatory security for the entire database system; because all of the database system software is untrusted with respect to mandatory security, the database system software need not be examined or evaluated for correct enforcement of mandatory security properties. Because there is no trusted component in the database system itself, the risk of disclosure of sensitive data is considerably reduced. This is the most conservative approach possible for mandatory security.

3.3 Polyinstantiation

SeaView introduced the concept of *polyinstantiation*, by which different versions of the same real-world entity can be represented in the database, where the different versions represent what is known to users at different clearance levels. Polyinstantiation has two fundamental forms. *Entity polyinstantiation* arises when a person with a low clearance assigns what is intended to be a unique identifier (for example, employee ID number) to a real-world entity (for example, a person) known to people with low clearances, and is unaware that the identifier has already been assigned to some other real-world entity known only to persons with high clearances. The result will be that there are two distinct real-world entities having the same "unique" identifier. For example, suppose a low user enters an employee with employee-ID number 12345, unaware of the fact that a high employee 12345 already exists. To preclude the possibility of an insecure information flow, the low person cannot be informed of the conflict. *Attribute polyinstantiation* arises when a person with a low clearance assigns a value to some attribute of a real-world entity known to persons with low clearances, when that real-world entity has in fact a more highly classified value for that attribute. For example, a low space shuttle flight could have the low mission "space-exploration" known to people with low clearances, and the high mission "spying" known to people with high clearances.

In a multilevel database system with element-level classification, polyinstantiation arises in two varieties: polyinstantiated tuples and polyinstantiated elements [4]. Polyinstantiated tuples represent entity polyinstantiation, whereas polyinstantiated elements represent attribute polyinstantiation. Polyinstantiated elements are represented by a set of tuples, all of which have the same primary key value and primary key classification. The SeaView prototype demonstrates the following aspects of polyinstantiation:

- *Polyinstantiated tuples* are identified by a primary key and associated key class, so that the same multilevel relation may contain several tuple instances for a primary key value corresponding to different access classes. A polyinstantiated tuple arises when a subject inserts a tuple that has the same primary key value as an existing but invisible (more highly classified) tuple. The effect of the operation is to add a *second* tuple to the relation, whose primary key is distinguishable from the first by its access class. Although the polyinstantiation is invisible to this subject, subjects at the higher access class can see both tuples.

- *Polyinstantiated elements* are identified by a primary key, key class, and element class (in addition to the attribute name), so that there may be multiple elements for an attribute that

have different access classes but are associated with the same (primary key, key class) pair. A polyinstantiated element arises when a subject updates what appears to be a null element in a tuple, but which actually hides data with a higher access class. In this case, the update has the effect of creating a polyinstantiated element for the tuple. A polyinstantiated element can also arise when a high subject updates a low element — instead of overwriting the low element value, a polyinstantiated element is created. In SeaView, polyinstantiated elements are represented as separate tuples.

- *High subjects cannot overwrite or replace low data.* Polyinstantiation prevents high subjects from overwriting or replacing low data; such behavior by high subjects would constitute a signaling channel. Instead, if a high subject attempts to update a low value, a polyinstantiated element is created. After the update, low subjects see the original, unchanged low tuple, and high subjects see *two* tuples, which are identical except for the value and classification of the polyinstantiated element; thus, high subjects can see both the high and low versions of the tuple.

- *Low subjects cannot overwrite or replace high data.* If a low subject attempts to update a null value that actually hides high data, SeaView does not allow the low subject to overwrite the high data but instead creates a polyinstantiated element. After the update, low subjects see the tuple updated with the new low value, and high subjects see *two* tuples, which are identical except for the value and classification of the polyinstantiated element; thus, high subjects can see both the high and low versions of the tuple.

- *Low subjects do not know of polyinstantiated high data.* Polyinstantiation is invisible to and undetectable by low subjects. High subjects can see multiple versions of tuples with polyinstantiated elements, and can see multiple polyinstantiated tuples, but low subjects see only low data. A subject can see only data at the subject class and below.

- *High subjects can choose the most appropriate versions with MSQL support.* Because high subjects can see multiple versions of polyinstantiated data, they can choose the version that is most appropriate for their purposes. The MSQL language provides support for selecting data based on their access classes and for selecting the highest, lowest, or most recent data from among polyinstantiated data. These functions can be embedded in applications so that high users need never even be aware of the polyinstantiation that exists.

- *Prevents information leakage.* Polyinstantiation is necessary in order to hide the actions of high subjects from low subjects, thereby preventing signaling channels. Polyinstantiation also prevents low users from inferring the existence of, or values of, high data. For example, if a low subject attempts to insert a tuple with an apparent primary key value equal to that of a preexisting high tuple, preventing the low user from inserting the tuple is tantamount to telling the low user the value of the high primary key. Continued attempts by low users to insert tuples could reveal the values of *all* the high primary keys. Polyinstantiation prevents this by allowing polyinstantiated tuples.

- *Allows implementation of cover stories.* Polyinstantiation can occur deliberately in the form of *cover stories*. A cover story is needed when some real-world entity is unavoidably visible to people with low clearances, but some attribute of that entity whose existence is known or can be assumed at the low level is classified higher than the entity itself. A cover story is used to give a plausible explanation and to prevent the guessing or inference of the classified attribute value. For example, if massive troop movements are known at the low level (because it is impractical to hide the fact) and the reason is not known to people with low clearances, these people may speculate or infer the true reason unless a plausible explanation is given. In this example, “military exercise” may be a cover story for “staging for battle.” Cover stories

may be considered deliberate polyinstantiation. They are necessary from a security point of view to prevent undesired inferences.

3.4 Decomposition and Materialization

SeaView implements multilevel relations as views over underlying single-level base relations [1]. This approach allows multilevel `select`, `insert`, `update`, and `delete` operations to be decomposed into corresponding single-level operations on the single-level base relations, and lends itself to a design that uses a commercially available RDBMS for the single-level relations. The decomposition is transparent to the user, who considers the multilevel relations to be the actual stored relations of the database system. The decomposition allows the underlying operating system's mandatory TCB to enforce the mandatory security policy on a set of single-level objects and thereby restrict the data visible through a multilevel relation to that dominated by the subject class; no filtering of high data from multilevel relations is required in the database system software. This approach lends itself to a high-assurance implementation if a high-assurance operating system is used.

The SeaView prototype demonstrates the following features and characteristics:

- *Decomposition of multilevel relations.* Multilevel user relations are transparently decomposed into single-level Trusted ORACLE relations according to a decomposition algorithm we developed for the prototype. When a user creates a multilevel relation, SeaView transparently creates a set of single-level Trusted ORACLE tables to hold the data that will later be inserted into the multilevel relation. In addition, SeaView transparently creates a set of views, one at each access class at which the multilevel table is visible, that defines the instances of the multilevel relation that are visible at the corresponding access classes.

- *MSQL queries translated into SQL queries.* The prototype includes an MSQL Processor that accepts user queries expressed in the MSQL multilevel query language and translates them into a set of SQL queries for Trusted ORACLE. Trusted ORACLE executes the queries and returns the results to the MSQL Processor.

- *SQL query results mapped back into answers to MSQL queries.* Trusted ORACLE returns the query results to the MSQL Processor, which assembles them into a query result appropriate to the original user query.

- *Transparent decomposition/materialization.* All user access must be made through the MSQL Processor; users have no access to an SQL interface. Thus, the system behaves just as if the user-defined multilevel relations were the actual stored relations in the multilevel database. The user is unaware of the decomposition.

- *Transparent tuple timestamp.* The prototype includes a transparent timestamp attribute for each tuple in a multilevel relation that is treated as part of the primary key and is stored in each underlying base table. This timestamp attribute is used when the single-level base tables are joined to materialize a multilevel relation. The use of the timestamp prevents newly inserted low data from being erroneously associated with "dangling" high data.

- *Little effect on query performance.* Our experience with the SeaView prototype is that the decomposition and materialization of multilevel relations does not significantly affect query performance.

- *Effect on performance of table creation.* Creating large tables with lots of access classes is much slower than creating correspondingly large tables in standard database systems. This is because the system must create a large number of Trusted ORACLE tables and views for a single MSQL `create table` statement. However, this is a one-time penalty incurred when

a multilevel relation is initially created; subsequent queries on the multilevel relation do not incur a similar performance penalty. Because users generally do not create tables (tables are designed and created by the applications builders), we feel this is a suitable tradeoff.

3.5 MSQL Query Language

SeaView presents users with the abstraction of multilevel relations with element-level classification. So that users can define and manipulate multilevel relations, we provide a multilevel query language, which we call *MSQL*. MSQL is an extension of SQL (Structured Query Language) and includes user commands for operating on multilevel data. MSQL is an implementation-independent language that makes no assumptions about the underlying system architecture. MSQL is meant to be upwardly compatible with SQL; that is, SQL programs and queries should also run on systems implementing MSQL. The SeaView prototype demonstrates the following features of MSQL:

- *Interactive MSQL language interface.* The prototype provides an interactive interface for users entering MSQL commands at the keyboard through a CMW window.

- *Forms interface.* The prototype also provides an interactive forms interface, which is implemented using SQL*FORMS (a product of Oracle Corporation). The forms interface allows a user to select data for update by using the mouse and to update data by direct overwriting. It also allows a user to formulate a query by traversing a set of menus and filling in the appropriate forms. A user need not know MSQL to be able to use this interface.

- *MSQL select, insert, update, and delete statements.* The prototype allows users to retrieve and manipulate multilevel data through the use of the MSQL `select`, `insert`, `update`, and `delete` statements. These statements can contain conditional clauses that select data for retrieval, update, or delete based on their access classes or on the values of operators on or functions over their access classes.

- *MSQL create table statement.* MSQL provides a `create table` statement that allows a user to define a multilevel relation. The statement requires the designation of a primary key, allows the designation of a foreign key (FK), and requires a designation for each attribute group of the range of access classes for data that can appear for attributes in the group.

- *MSQL support for managing polyinstantiation.* MSQL includes a number of features to aid the user and the data designer in limiting and managing polyinstantiation. The MSQL `create table` statement allows uniformly classified attributes to be grouped and to thereby share a common label within any tuple. This allows the SeaView decomposition algorithm to treat each such group as a single attribute, thereby reducing the number of underlying base tables that must be created. It is perhaps more significant that the grouping of attributes means that they cannot be independently polyinstantiated, since they must always share a common label within any single tuple. In addition, the MSQL `create table` statement requires that the user designate, for each attribute or uniformly classified attribute group, a range of allowable access classes. By using single-level ranges for attributes and attribute groups, the data designer can eliminate the possibility of polyinstantiation within such attributes and attribute groups. By specifying narrow ranges, the data designer can limit the amount of polyinstantiation that can occur. MSQL also includes direct language support for selecting particular instances from among polyinstantiated data. MSQL provides functions and operators for selecting data based on access class and for selecting the highest, lowest, or most recent data from among polyinstantiated data. These functions can be embedded in applications so that users need not be aware of the polyinstantiation that exists.

- *MSQL data dictionary.* The prototype includes an MSQL data dictionary that stores the definitions of multilevel user relations as well as the information needed to map a multilevel relation to its implementing single-level Trusted ORACLE tables and views. The portion of the MSQL data dictionary that describes multilevel relations is user-visible; the portion that defines the mapping to the underlying base tables and views is not accessible by users. MSQL data dictionary entries are made automatically by the execution of MSQL `create table` statements.

4 Characteristics of the SeaView Prototype

We have evaluated the SeaView prototype for performance, software size, and portability.

The performance of a database system can be evaluated in various ways. One of the more popular techniques uses a set of predefined relational database tables and views loaded with benchmark data to measure transactions per second (TPS). Typically, this TPS benchmark serves two purposes. The first purpose is to compare the performance of the database engine running on different hardware platforms. The second purpose is to compare different database products using the same data model running on the same hardware platform. SeaView, however, cannot benefit from either performance measurement category because there are no competitive prototypes or commercial products which support the same data model as SeaView. In addition, the SeaView prototype is available only on the Sun CMW platform. For these reasons, it is difficult for us to establish a comparison basis and conduct an objective and impartial performance measurement.

It is conceivable that a typical SeaView user table with three or four attributes may require joining some twenty or more base tables and views, depending on the number of labels associated with each column. Because joining multiple tables or views is expensive, SeaView queries are potentially expensive. To speed query operations, we have defined indices on the join attributes in our decomposition algorithm. These indices expedite the join operations with an indexed search, rather than with a time consuming sequential search. Although the SeaView system incurs a performance penalty for this decomposition and rematerialization, other database systems running similar applications will require the application to perform the decomposition and rematerialization of multilevel data, and hence to incur the performance penalty. Thus, a fair performance comparison of SeaView with row- or table-labeled database systems should include the implementation of multilevel applications.

We measured the performance of the SeaView prototype with varying amounts of stored data and degrees of complexity of multilevel tables so that we can quantify and extrapolate performance statistics. The outcome of the performance analysis was encouraging. It is our observation that the SeaView prototype software incurs no perceivable performance cost beyond that required to join the underlying base tables and views — that is, the cost incurred by the SeaView software is imperceptible to users.

A typical multilevel table is composed of several base tables and views. The SeaView table creation statement consumes considerable processing time because Trusted ORACLE needs that much time to create all the base tables and views. To assist the data designer in controlling this performance cost, SeaView includes features, such as the ability to specify uniformly classified groups of attributes, and the ability to specify a range of allowable classifications for each attribute group, to improve table creation performance.

The SeaView MSQL Processor contains 26,303 lines of C code, an estimated 5% of which is comments. We have used ORACLE Pro*C to preprocess the embedded SQL statements and

the GNU C compiler in our development environment to compile the C code on the Sun CMW.

The SeaView prototype contains only six ORACLE-dependent Pro*C modules and one Sun CMW label-specific module. Thus, it should be relatively straightforward to port SeaView to other secure platforms. For example, we spent about three months in porting SeaView from GEMSOS to the Sun CMW. We estimate that future porting of SeaView to other comparable secure platforms that support Trusted ORACLE would take a comparable amount of time.

5 SeaView Prototype Enhancements

Several enhancements would facilitate the transition of SeaView into operational use. SeaView could be ported to high-assurance platforms such as the HFSI XTS-200 or the SCC LOCK. Under a Rome Laboratory contract, Trusted ORACLE is being ported to LOCK/SNS, a Class A1-evaluatable TCB. That port is expected to be completed in the second half of 1994. When that work is completed, SeaView can also be ported to that Class A1 platform. This would allow a high-assurance implementation of SeaView for little additional cost.

In addition, our preliminary design for SeaView's DAC feature could be implemented. A set of programatic calling services to allow development of third-party applications on top of the SeaView system could be implemented (some of this is already being done under a separate project for which the first application is being built on SeaView [19]). SeaView could be upgraded to support increased pass-through of Trusted ORACLE functionality. MSQL maintenance utilities could be implemented that would allow users to perform various database maintenance operations such as data import/export, load/unload, database backup/restore, and garbage collection. Additional sample applications could be built on SeaView (A prototype multilevel X.500 DoD Directory service has already been implemented on SeaView by SRI [19].) SeaView's decomposition algorithm could be revised to improve performance and reduce the complexity in retrieving multilevel polyinstantiated data, to incorporate modifications suggested by Jajodia and others [20]. Future work on SeaView may include the design and implementation of facilities to support classification constraints that would assign classification labels to data entered into the database. Tools could be included in SeaView to assist the data designer in anticipating and limiting the amount of polyinstantiation that can occur. Our current work to build a prototype inference control tool [21, 22] could be integrated into SeaView to provide a graphical front-end user interface for design of an inference-free SeaView database schema and automatic generation of the database schema definition.

6 Summary

The SeaView multilevel database prototype is an ambitious project that attempts to meet the requirements for Class A1 of the U.S. Orange Book for a system that features data classification at the granularity of individual data elements. We have described key aspects of the SeaView model and reported on our implementation of a prototype multilevel database system based on the SeaView model.

The SeaView prototype proves the concept of the original SeaView model based on strict TCB subsetting. It demonstrates and proves the concept of polyinstantiation. It uses a modular and layered architecture that makes use of the commercial-off-the-shelf Trusted ORACLE database system; this allowed SeaView developers to focus on designing and implementing

critical functionalities such as a multilevel decomposition algorithm and a multilevel data dictionary. The prototype includes a functioning MSQL query language and features. Because it is implemented on Trusted ORACLE, SeaView can be readily ported to other platforms. Porting to a high-assurance platform will result in an A1-evaluatable implementation. Based on this SeaView prototype, we believe that it is possible to develop a high-assurance, production-quality, multilevel secure relational database system based on the SeaView model for real-world use in the near future.

References

- [1] D. Hsieh, T. F. Lunt, and P. K. Boucher. *The SeaView Prototype Final Report*. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, August 1993.
- [2] T. F. Lunt, D. E. Denning, P. G. Neumann, R. R. Schell, M. Heckman, and W. R. Shockley. *Final Report Vol. 1: Security Policy and Policy Interpretation for a Class A1 Multilevel Secure Relational Database System*. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 1988.
- [3] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley. *Final Report Col. 2: The SeaView Formal Security Policy Model*. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 1989.
- [4] T. F. Lunt, D. E. Denning, R. R. Schell, W. R. Shockley, and M. Heckman. The SeaView security model. *IEEE Transactions on Software Engineering*, June 1990.
- [5] T. F. Lunt. *Final Report Vol. 4: Secure Distributed Data Views: Identification of Deficiencies and Directions for Future Research*. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 1989.
- [6] W. R. Shockley, R. R. Schell, T. F. Lunt, D. Warren, and M. Heckman. *Final Report Vol. 5: The SeaView Implementation Specifications*. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 1989.
- [7] Department of Defense. *Trusted Computer System Evaluation Criteria, DOD 5200.28-STD*, December 1985.
- [8] National 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*, June 1985.
- [9] Secure Computing Corporation. *System Specification for the LOCK DBMS Program*. Secure Computing Corporation, Roseville, Minnesota, 1993.
- [10] T. H. Hinke, C. Garvey, N. Jensen, J. Wilson, and A. Wu. A1 secure DBMS design. In *Proceedings of the 11th National Computer Security Conference - Appendix*, October 1988.
- [11] National Computer Security Center. *National Computer Security Center Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*. Technical Report NCSC-TG-021, Version 1, April 1991.

- [12] W. R. Shockley and R. R. Schell. TCB subsetting for incremental evaluation. In *Proceedings of the Third AIAA Conference on Computer Security*, December 1987.
- [13] M. Schaefer and R. R. Schell. Toward an understanding of extensible architectures for evaluated trusted computer system products. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, April 1984.
- [14] T. F. Lunt and R. A. Whitehurst. *Final Report Vol. 3a: The SeaView Formal Top Level Specifications*. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 1989.
- [15] R. A. Whitehurst and T. F. Lunt. *Final Report Vol. 3b: The SeaView Formal Verification: Proofs*. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 1989.
- [16] F. von Henke and J. Rushby. *Introduction to EHDM*. Computer Science Laboratory, SRI International, Menlo Park, California, September, 1988.
- [17] P. K. Boucher and T. F. Lunt. *The SeaView Formal Top-Level Specification to Code Correspondence*. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 1993.
- [18] R. R. Schell and D. E. Denning. Integrity in trusted database systems. In *Proceedings of the 9th National Computer Security Conference*, 1986.
- [19] P. K. Boucher and T. F. Lunt. A prototype multilevel-secure DoD Directory. Submitted to the Tenth Annual Computer Security Applications Conference.
- [20] S. Jajodia and R. Sandhu. Polyinstantiation integrity in multilevel relations. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, May, 1990.
- [21] X. Qian, M. E. Stickel, P. D. Karp, T. F. Lunt, and T. D. Garvey. Detection and elimination of inference channels in multilevel relational databases. In *Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy*, May 1993.
- [22] M. E. Stickel, X. Qian, T. F. Lunt, and T. D. Garvey. *Inference Channel Detection and Elimination (second interim report)*. Computer Science Laboratory, SRI International, Menlo Park, California, September 1993.

TOWARDS A FORMAL VERIFICATION OF A SECURE AND DISTRIBUTED SYSTEM AND ITS APPLICATIONS¹

Cui Zhang, Rob Shaw, Mark R. Heckman, Gregory D. Benson,
Myla Archer, Karl Levitt, and Ronald A. Olsson

*Department of Computer Science
University of California, Davis, CA 95616
Email: Last-Name@cs.ucdavis.edu*

Abstract

This paper presents research towards the formal specification and verification of a secure distributed system and secure application programs that run on it. We refer to the whole system — from hardware to application programs written in a concurrent programming language — as the Silo, and to a simplified view of the Silo as the miniSilo. Both miniSilo and Silo consist of a collection of microprocessors interconnected by a network, a distributed operating system and a compiler for a distributed programming language. Our goal is to verify the full Silo by mechanized layered formal proof using the higher order logic theorem proving system HOL. This paper describes our current results for verifying the miniSilo and our incremental approach for evolving the verification of the miniSilo into the verification of the full Silo. Scalability is addressed in part by extending the distributed operating system with additional servers which in turn provide services that extend the programming language.

Keywords: verification, distributed operating systems, security servers, distributed programming languages.

1 Introduction

This paper describes our research on a long term project called the Silo. This project is aimed at verifying a complete distributed computer system by mechanized layered formal proof. Our layered system includes a set of microprocessors, a network model, the operating system kernel and servers (some in support of security) running on each microprocessor (hence, a secure distributed operating system), the concurrent programming language microSR (a derivative of SR [1]), and a Hoare-like programming logic. Each layer will be formally modeled as an interpreter that interacts with the other layers. Our layered approach will allow us to verify that secure and distributed applications run correctly on the entire system. In its final form, the Silo will be somewhat limited when compared to “real” computer systems; however, we hope it will be the most comprehensive distributed computer system that is verified and demonstrates a methodology for “full system verification” of distributed systems.

The CLI stack [2] has shown the feasibility of full system verification for a sequential system using a layered proof technique, but their model does not allow for concurrency and distributed programming, nor have they fully integrated the operating system into their “stack”. When we began specifying the Silo system, we realized that an incremental approach is necessary for revealing unforeseen difficulties and for making the formal proof more manageable. Rather than attempting to specify and prove the entire Silo, we have identified a subset of the Silo to specify and prove correct by limiting the scope of each layer to reduce the complexity. As shown in Figure 1, we refer to this simplified view of the Silo as the miniSilo. As our preliminary results on miniSilo have

¹This work was sponsored by the National Security Agency University Research Program under contract DOD-MDA904-93-C-4088 and by ARPA under contract USN N00014-93-1-1322 with the Office of Naval Research.

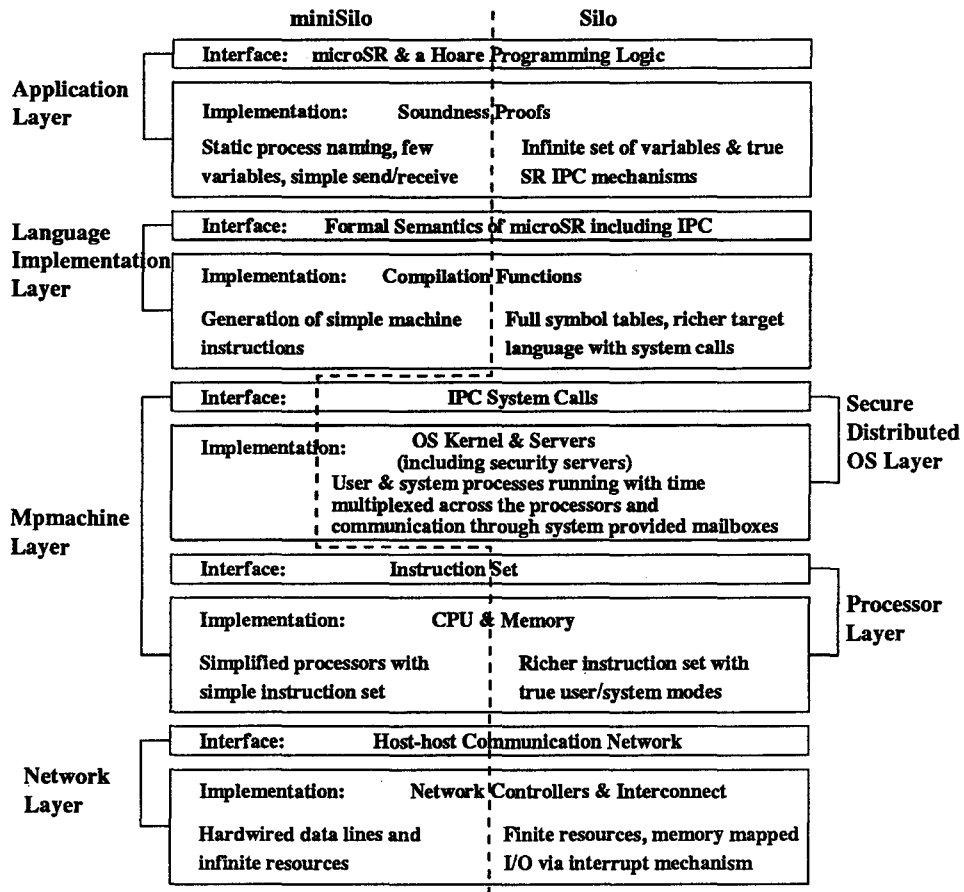


Figure 1: Overview of UCD Silo and MiniSilo

shown the usefulness of our layered proof methodology, we are now growing the miniSilo system into the full Silo by developing the system and proof by incrementally adding functionality to all layers. Our specification, verification, and augmentation process is being carried out using the Cambridge HOL theorem prover [8], because it allows the definition of embedded theories, such as we are using for a programming logic of concurrency and a generic model of a layer. We also hope our work demonstrates the expressiveness, flexibility, and feasibility of higher order logic in formal specification and verification for more complicated computer systems, including a concurrent programming language that support security applications and a distributed operating system.

This paper concentrates on our miniSilo effort, as a step in the full Silo effort. Section 2 describes our work on the network layer. Section 3 gives our work on the mpmachine layer. Section 4 describes our effort on the language implementation for microSR. Section 5 presents the Hoare logic derived from the microSR semantic specification. Section 6 concludes our work.

2 The Network

2.1 The Network for MiniSilo

The lowest layer of miniSilo consists of a network which allows individual processors (vmachines, see Section 3) to communicate through message passing. The miniSilo network consists of a set of processors and an interconnect service. Each processor communicates with the network through a

Network Interface Unit (NIU), as shown in Figure 2. In miniSilo we assume that a processor has dedicated, hardwired data lines that interface directly with an NIU. The network provides reliable transmission of messages and preserves message ordering between communicating processors.

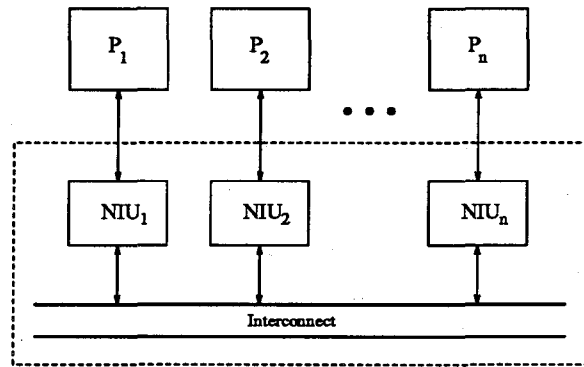


Figure 2: The MiniSilo Network

The miniSilo network is specified abstractly. By specifying the network in general terms, we do not impose any restrictions on the network topology or on the communication protocol. We do ensure that the network provides the properties that the higher miniSilo layers assume of the network. Later, if desired, one could develop an implementation of the abstract network specification. The next logical layer is the protocol layer. There has been considerable work on the verification of network protocols [5, 12], which could be used to implement the abstract specification presented. For “complete” verification the protocol layer must ultimately be specified in terms of the underlying hardware. Protocol and network hardware verification are beyond the scope of this project. The network is also specified operationally, where each NIU is modeled as an interpreter that reads and modifies state. The entire network is modeled as the composition of all the NIUs. The network interpreter is driven by send requests from the processors. Send requests result in receive requests from an NIU to a processor, which allows for nonblocking I/O at the operating system level. Sends and receives are accomplished through memory mapped I/O.

The set of NIUs are modeled as a fully connected network through send and receive queues, collectively called in-transit queues. For n processors, each NIU has $n - 1$ send queues and $n - 1$ receive queues. Each queue is shared by exactly two NIUs, one NIU views the queue as a send queue and the other NIU views it as a receive queue. The send and receive queues form the *InTransit_State*. The *NIU_State* for each NIU combined with the *Intransit_State* form the *Network_State*. The specification of the network interpreter is a relation, $Network_State \rightarrow Network_State \rightarrow Bool$. This interpreter is used to prove properties about the network itself, as well as to serve as an implementation for the higher layers of miniSilo.

Because the miniSilo network specification is given in terms of abstract operational semantics, we need to prove certain safety properties to ensure that the network functions correctly. The most important safety property is the ordering of messages between communicating processors. This property follows from the representation of the *InTransit_State*. Other safety properties, such as no duplication of messages, are also verified.

2.2 The Network for Silo

The proof obligation of the mpmachine requires us to verify that the network specification combined with the vmachine specification logically imply the mpmachine specification. In miniSilo, the distinction between the mpmachine communication abstractions and the network abstractions are

small, but this will change once the full Silo is developed and each layer is expanded to more realistic specifications of a distributed system. In particular, the network specification will be modified in two respects. First, the network will be specified in terms of finite resources rather than infinite resources. Currently the specification allows infinitely many messages to be present in the in-transit queues. Therefore, each NIU is always ready to send another message, the processor is not required to wait or resend messages. Moving from infinite queues to finite queues entails certain specified error conditions and can result in storage channels. In miniSilo, we also assume that the message being transferred is a single, but infinite integer. We intend to alter the specification to handle finite packets. Second, the interface between an NIU and a vmachine will be enhanced to one based on memory-mapped I/O and interrupts rather than memory-mapped I/O alone. This will allow the operating system to implement non-blocking I/O, and more importantly, allow for more than one process and operation per processor as described in Section 3. The new processor to NIU interface will also be enhanced to handle simple error conditions such as a network busy error or packet lost error, both of which will result in the processor resending the packet. Again, there are security implications to these decisions, which we will consider.

3 The Mpmachine

3.1 The MiniSilo Mpmachine

The miniSilo abstraction mpmachine represents multiple processors, each running a single process. Processes communicate by passing messages through a network. From a user process's point of view, the operating system interface appears as an "extended machine", consisting of the basic machine instructions plus communication primitives (system calls). The communication primitives are used to send and receive messages, through message queues. MiniSilo has one message queue per process, where only one process can read from the queue and all other processes can send messages to the queue.

The vmachine specification describes a single processor in miniSilo. Each vmachine² consists of an infinite set of registers, an infinite set of memory locations, and a program counter. Since these are modeled in HOL using natural numbers, each location may hold a non-negative integer of any size. A single vmachine operates much as one would expect, interpreting a typical set of simple machine instructions consisting of load, store, arithmetic, comparison, and branching instructions. It can not, however, issue any kind of communication action with other vmachines; the mpmachine provides this ability. This modularization is intended to isolate the processor from changes in the network hardware — the mpmachine is responsible for the compatibility of these two lower components and for defining the pool of message queues and system calls. Neither component depends upon the other's specification in any way.

An mpmachine contains N vmachine processors and N network interface units (NIUs) connected to a bus. Within the mpmachine specification, however, this bus is abstracted as a pool of queues. This pool contains one queue for each NIU, representing the ordered list of pending messages destined for the vmachine corresponding to the particular NIU. The external appearance of an mpmachine, therefore, is an N -tuple of vmachines (whose appearance is "passed-up", unaltered), plus a pool of "in-transit" message queues.

Similarly, the language interpreted by an mpmachine is an N -tuple of lists of instructions. The set of instructions contains all the operations executable on a vmachine, plus communication primitives. Similar to earlier efforts [4, 10], the actual operation of the mpmachine is modelled with

²Initially, we chose this term as an abbreviation of "virtual machine". Presently, however, "vanilla machine" is perhaps more appropriate

transition relations. Each kind of transition allows a single component of the N -tuple to advance a single step. To issue a vmachine instruction, only the state of the corresponding vmachine hardware is affected. To issue a communication primitive, however, the global pool of queues may be altered as well.

The HOL specification of this machine model consists of straightforward type definitions for the objects described, plus the transition relation associated for each kind of mpmachine instruction. These relations have the type $Args \rightarrow MPprocess \rightarrow MPprocess \rightarrow Vid \rightarrow Bool$. The type $Args$ characterizes the numerical operands to the instruction. An $MPprocess$ represents a pair whose first component is the local state of the vmachine which is executing this instruction, and whose second component is the pool of queues. Finally, Vid is the index of the executing vmachine; this information is not available within an $MPprocess$. If we were to include, say, a read-only "processor id register" in each vmachine, then the information in Vid above would become redundant.

From this type definition, we see that the following question can be answered of each mpmachine instruction: Given the indicated operands, and the indicated initial configuration of the mpmachine, is it possible to arrive in a given configuration after the indicated processor executes this instruction? For example, the relation for a simple vmachine jump instruction would require that the pools in both $MPprocess$ objects are identical, because a jump does not affect communications. Moreover, the underlying vmachine specification would ensure that the register and memory contents of the vmachine object within the first $MPprocess$ must also be identical to the corresponding vmachine within the second $MPprocess$. Only the processor's program counter will differ between the two configurations, and this difference must agree with the target location given in the operand to the jump (indicated in the $Args$). The mpmachine specification does not directly contain these facts, but rather defers to the vmachine specification itself. As an example of message passing, if the instruction in question were a receive operation, both the processor and the pool contents will differ accordingly. In particular, after the instruction is complete, the destination register in the processor will contain the received value, and the appropriate queue in the pool (indicated by the Vid) will have one less message than it did before the instruction began.

Armed with a semantic relation for each instruction, the mpmachine specification only requires two more definitions to encompass the complete system behavior. The first of these, is an inductive definition of how a thread, an instance of a sequential program piece, may legally execute for $k \geq 0$ steps. To execute for zero steps, both the initial and the final $MPprocess$ must be completely identical. To execute for $k > 0$ steps, there must exist an intervening $MPprocess$ value, call it M , such that the appropriate semantic relation allows a one-step transition from the initial state into M , and the final $(k - 1)$ -step transition from M into the final state is allowed inductively. The second definition describes the legal behaviors of complete programs on an mpmachine, and it is not inductive. Here, a final state of the entire system is reachable from an initial one precisely when the corresponding initial and final $MPprocess$'s for each component of the program are allowed by the above inductive definition, for some $k \geq 0$.

3.2 Growth to Complete Silo

The complete Silo system consists of multiple processors, connected by a network and each running a copy of the Silo operating system. The operating system design is based on the kernel and server model used, for example, in Mach [14] and in Synergy [15]. The kernel provides a multi-programmed, message passing environment for the server processes and user processes on a particular processor. The abstraction of a distributed system is maintained by the servers. As shown in Figure 3, from a user process's point of view, the operating system interface in Silo will extend that of miniSilo with richer basic machine instructions and system calls. In this way, the

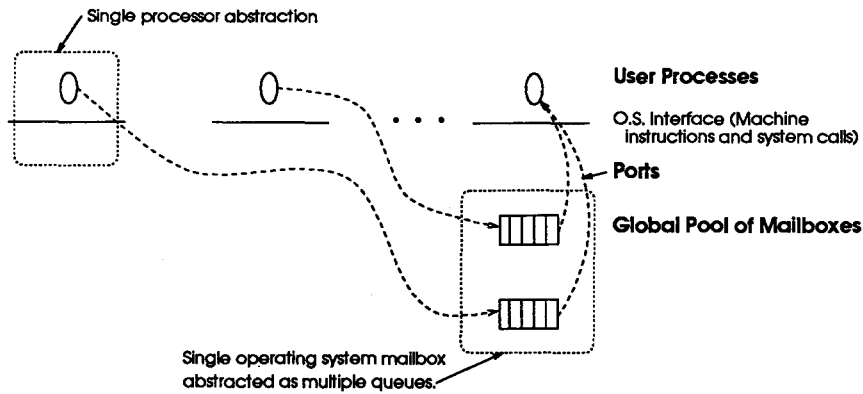


Figure 3: Operating System Specification: View from User Process

language work can proceed concurrently with the operating system work. Silo includes additional system calls for processes to create message queues, called *mailboxes*, and for processes to request access to specific mailboxes. The mailbox management calls are subject to a system security policy implemented by a security server, as shown in Figure 4. These calls, while an essential part of the Silo system specification, are only relevant to user processes when an application is initially loaded and, therefore, do not require significant changes to our language work.

A mailbox is a queue of messages with at most one process receiving messages through the mailbox and possibly many senders. The complete operating system specification guarantees that messages sent by a particular process to the same mailbox will be queued in the mailbox in the same order that they were sent but, due to the concurrent nature of the system, does not guarantee the relative ordering of messages sent by different processes. For this reason, Silo specifies a mailbox as a set of queues – one per sender, rather than one per receiver as in miniSilo.

A major challenge in the Silo project is to specify the entire distributed operating system at its interface to user processes, to specify each of the servers and the kernel, and to prove that a composition of the server, kernel and network specifications satisfy the secure distributed operating system specification. We are accomplishing this in stages: first composing the servers that manage mailboxes, then adding the servers that implement system security and support the security features in the programming language.

4 Implementation of MicroSR

4.1 MicroSR Semantics

The interpreted language at this layer is microSR whose constructs include those basic to common sequential programming languages, in addition to an asynchronous *send* statement, a synchronous *receive* statement, a guarded communication *input* statement, and a *co* statement for specifying concurrent execution. This language has the appearance of a high-level system programming language that supports distributed applications. For each statement, we have a semantic transition relation of type $Gstate \rightarrow Gstate \rightarrow Pid \rightarrow Bool$. These semantic relations are analogous to, though more complex than, the mpmachine relations. Here, the type *Gstate* (for “global state”) represents a complete system configuration, and the relation is true if and only if the system may evolve from the first *Gstate* into the second *Gstate* by the execution of the given microSR statement within the logical process indicated by *Pid*. The semantics are also formalized operationally, using multiple copies of a local state abstraction conjoined with a shared pool of messages. These

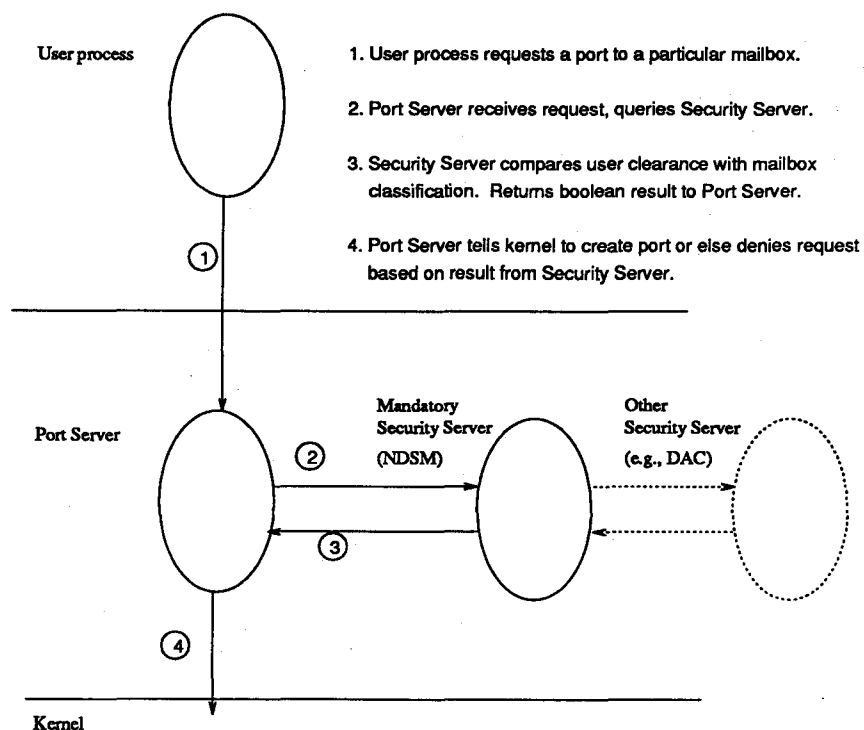


Figure 4: Operating System: Security management

local states are now mappings from variable names into values, rather than register and memory contents. However, the internal structure of this microSR message pool is almost identical to that of the mpmachine — for each program thread, the pool contains a queue of all messages which have been sent to this thread, plus an indication of which ones have been received thus far. To handle security, processes and objects are assigned security levels, and transitions are allowed if they satisfy the standard multilevel security policy.

4.2 Compiler Correctness

Like the previous successful efforts to prove compiler correctness for sequential languages [6, 9], to claim that a compiler is correct is to claim that the target code behavior achieve the source code semantics. Yet, as we have seen, the mpmachine behaviors and the microSR semantics are distinct enough that no canonical equivalence exists between them. We, as the verifiers, must provide this mapping from the abstract microSR global states down into the more concrete mpmachine states. As shown in Figure 5, once this mapping is available, the compiler correctness proof becomes an equivalence proof of two relations, given by the dashed line and the dotted line. For any given starting state, S , of the microSR program, these two relations must agree on which final mpmachine configurations are reachable. In particular, the compiler correctness condition is the following logical equivalence: If, the microSR semantics for the source program indicate that a certain final state, F , is reachable, then it must be true that the mpmachine semantics for the compiler's output code indicate that $F' = \text{Mapdown}(F)$ is reachable from $S' = \text{Mapdown}(S)$. The compiler itself is simply another mapping function over the domain of legal microSR programs that provides a list of mpmachine instructions for each construct.

A few implementation details are not evident in Figure 5. First of all, since both the Mapdown function and the compiler assign variables to registers, these two assignments must agree in order for

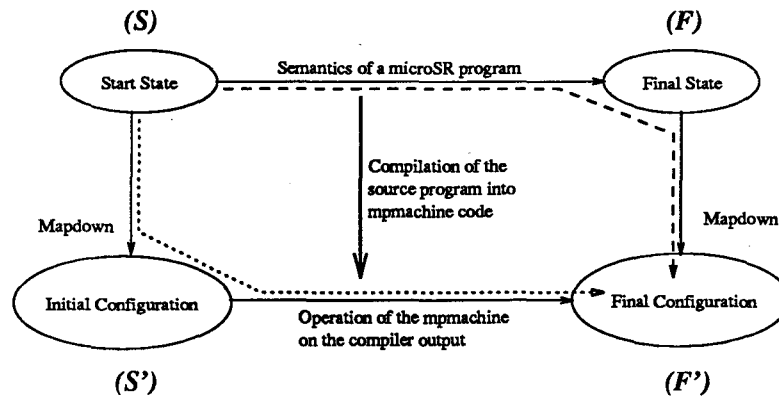


Figure 5: Necessary Mappings for Compiler Correctness

the above equivalence to hold. Consequently, the Mapdown function takes a symbol table argument that indicates the compiler's choices. In miniSilo, there is a fixed symbol table because the microSR language has a small, fixed set of legal variable identifiers. To allow arbitrary strings as identifiers, the compiler needs simply to make an initial pass over the source and gather the necessary symbol table information needed by Mapdown and the second pass. This process involves no concurrency nor composability issues whatsoever other than requiring the extra argument to Mapdown – an aspect that has been accommodated.

As described above, the “dashed” relation, whether true or false, must be equal to the “dotted” relation. This is not entirely possible because a small amount of information is lost across the Mapdown function. For instance, a microSR global state contains a component that indicates the current time of the state. Suppose that we have two states, S_1 and S_2 , which are legal starting and ending states for some program. Both the dashed and dotted relations indicate truth. However, suppose that we now alter S_2 ever so slightly, by making its time indicator earlier than that of S_1 . Since the global time does not appear in the mpmachine specification, the result of mapping down S_2 is just as it was before, and the dotted relation continues to indicate truth. The dashed relation, however, does not allow for time to decrease, and indicates falsity. The use of the time counter is merely an example; the microSR semantics contain other auxiliary data, such as the number of receives on a particular channel, that are not mapped down to the hardware level. Indeed, when the full Silo contains a kernel with many internal tables, it would not even be clear how the language-level receive counts should be mapped. We do not want the language layer imposing bookkeeping requirements on the kernel, and the correct choice is to not map down the information that is only needed by the language semantics. As a result, the compiler proof is not a complete equivalence, but it must distinguish different means by which the language semantics may indicate falsity.

Similarly, the mpmachine abstraction also contains some items that are not within the image of Mapdown. The first few memory locations are considered to be “reserved” for system use, and the Mapdown function does not dictate the values of the addresses. The fact that the language layer relation holds does not impart any knowledge about this hidden system state within the mpmachine. Consequently, the actual proof requires a third machine configuration (not shown) which is both reachable from S' and equivalent to F' in all respects except the hidden system state. Finally, within this proof, the complete program is really viewed as a collection of processes, and the picture indicates what must be shown for each individual process. Rather than use fully defined states and configurations, we show that for each process, the relationships of Figure 5 hold amongst that process' view of the system state.

5 MicroSR Applications

5.1 The Hoare Logic for MicroSR

The top application layer is a mechanized Hoare logic for verifying microSR concurrent applications. Our effort to formally derive, using HOL, a sound Hoare logic from microSR semantics is a generalization of similar work by Gordon for a small sequential language [7, 13]. We use semantic relations, rather than functions, in our formal specification for microSR constructs; doing so obviates the possible need for powerdomains in the state abstraction for microSR programs due to the inherent non-determinism. To handle the interference problem arising from concurrent execution, We introduced atomicity and global invariants [2] into our logic system. This logic has been formally proven to be sound within HOL, i.e., axioms and inference rules are all mechanically derived in HOL as the logical implication of the same microSR semantic specification against which the microSR implementation is verified. This logic allows one to reason and state formal assertions about concurrently executing processes that do not share any data objects, but communicate through shared channels that are called operations in SR terminology.

The partial correctness specification in our logic has two levels. The definition of predicate SPEC shown below gives our interpretation of $\{P_and/or_GI\} S \{Q_and/or_GI\}$, the intra-process partial correctness specification, where S is the microSR statement, P and Q are assertions mainly on program variables, GI is the assertion of global invariant mainly on operations, associated with executing S and taken with respect to a particular process. The definition of predicate G_SPEC gives our interpretation of the global partial correctness specification $\{(P_list) \wedge GI\} S \{(Q_list) \wedge GI\}$, where S is the top level statement for specifying concurrent executions, global invariant GI is the assertion mainly on operations, P_list and Q_list are assertion lists mainly on program variables. The i th elements of the two lists are taken with respect to a particular process for executing the i th sequential program within the top level statement S . Notice that all arguments of SPEC and G_SPEC in the following definitions are abbreviated forms of their meaning functions.

$$\begin{aligned} \text{SPEC } (P_and/or_GI, S, Q_and/or_GI) &= \vdash \text{def } \forall \text{ Gstate1 Gstate2 Pid .} \\ &P_and/or_GI(\text{Gstate}, \text{Pid}) \wedge S(\text{Gstate1}, \text{Gstate2}, \text{Pid}) \\ &\Rightarrow Q_and/or_GI(\text{Gstate2}, \text{Pid}) \\ \text{G_SPEC } ((P_list) \wedge GI, S, (Q_list) \wedge GI) &= \vdash \text{def } \forall \text{ Gstate1 Gstate2 Pid_list .} \\ &(\forall i . (\text{El } i \text{ P_list})(\text{Gstate1}, (\text{El } i \text{ Pid_list})) \wedge GI(\text{Gstate}, (\text{EL } i \text{ Pid_list}))) \wedge \\ &S(\text{Gstate1}, \text{Gstate2}, \text{Pid_list}) \\ &\Rightarrow (\forall i . (\text{El } i \text{ Q_list})(\text{Gstate2}, (\text{El } i \text{ Pid_list})) \wedge GI(\text{Gstate2}, (\text{EL } i \text{ Pid_list}))) \end{aligned}$$

The following gives our representative axioms and inference rules in the derived logic for microSR. Those axioms and rules for microSR sequential constructs, such as the Skip Axioms, Assignment Axiom, If Rule, Do Rule, Sequencing Rule, Precondition Strengthening Rule, and Postcondition Weakening Rule, are not listed below, because their appearance is similar to that in [2, 7], though the way to formally specify and derive them for microSR is actually more complex. All axioms and inference rules are theorems of our language semantics. The “sent-set” σ and “received-set” ρ denote all messages ever sent and received on that channel. $\text{Frontier}(\sigma_{op})$ denotes the earliest message in the channel op that has not been received. μ is simply a message constructor function for converting an entity of type integer into one of type message.

- Co Rule

$$\frac{\{GI \wedge Pi\} \text{SLi } \{GI \wedge Qi\}}{\{\{GI \wedge P_list\}\} \text{co SL1 // ... // SLn oc } \{\{GI \wedge Q_list\}\}}$$

- Send Axiom

$$\{P \wedge GI \wedge GI_{\sigma_{op} \cup \mu(E)}^{\sigma_{op}}\} \text{send } op \ (E) \ \{P \wedge GI\}$$

- Receive Rule

$$\frac{P \wedge GI \wedge \mu(E) \in \text{Frontier}(\sigma_{\text{op}}) \Rightarrow Q_E^v \wedge GI_{\rho_{\text{op}} \cup \mu(E)}^{\rho_{\text{op}}}}{\{P \wedge GI\} \text{ receive op}(v) \{Q \wedge GI\}}$$

- In Rule

$$\frac{\{P \wedge GI\} \text{ receive op1}(v) \{R1 \wedge GI\} S1 \{Q \wedge GI\}, \{P \wedge GI\} \text{ receive op2}(v) \{R2 \wedge GI\} S2 \{Q \wedge GI\}}{\{P \wedge GI\} \text{ in op1}(v) \rightarrow S1 \square \text{ op2}(v) \rightarrow S2 \text{ ni } \{Q \wedge GI\}}$$

5.2 Extensions for Silo

Following our incremental approach, we expect that our final language for Silo will be close to its parent language in its expressive power for distributed computing and our logic will be extended as well. For instance, in our current version of microSR, input statements support only message passing because operations serviced by an input statement can only be invoked by send statements. In our later version, we will allow operations to be invoked by call statements, which will provide rendezvous. We will also extend our input statement with synchronization expressions to allow selective receipt. We will also add some feature into our language to allow users to specify the security level of their programs, resources and processes that they create. The current results at this layer serve as a basis of our research for the complete Silo, since our research so far indicates that SR concurrency features, such as dynamic process creation and that synchronization via message-passing, remote procedure calls, and rendezvous, are all amenable to a Hoare-like programming logic, because the components of our semantic model for microSR have already formalized most of entities and behaviors that SR programmers must consider during their design process. We are now also evaluating the expressive power of our logic by carrying out proofs of programs. The preliminary attempts at manual proof of microSR programs have motivated us to establish a systematic method for creating annotated microSR programs. Another challenging task is to develop, using HOL as well, an interactive prover of LCF [11] style for microSR.

6 Conclusion

Our research on miniSilo has shown how to structure proofs according to vertical layers, how to formally model different layers, how to model the interactions between layers, how to express the proof obligations between layers, and how to compose all the proved layers together. We are extending our research on system design and proof to show that how to evolve miniSilo to Silo in an incremental manner. By our layered proof, we hope to demonstrate that secure and distributed applications can be verified with respect to the entire system, namely showing that microSR applications that are proved correct in our Hoare logic will run correctly on our Silo system.

References

- [1] G.R. Andrews, R.A. Olsson, M. Coffin, I.J.P. Elshoff, K. Nilsen, T. Purdin, and G. Townsend, An Overview of the SR Language and Implementation, ACM Transactions on Programming Languages and Systems, 10 (1988) 51-86.
- [2] G.R. Andrews, Concurrent Programming: Principles and Practice, The Benjamin/Cummings Publishing Company, Inc. Redwood City, CA, 1991.
- [3] W.R. Bevier, W.A. Hunt, J.S. Moore, and W.D. Young, An approach to systems verification, Journal of Automated Reasoning, 5 (1989) 411-428.

- [4] W.R. Bevier, and J. Sogaard-Andersen, Mechanically Checked Proofs of Kernel Specifications, in CAV '91, number 575 in Lecture Notes in Computer Science, pp.70-82, Springer Verlag, July 1991.
- [5] R. Cardell-Oliver, Using Higher Order Logic for Modelling Real-time Protocols, in TAPSOFT '91, number 494 in Lecture Notes in Computer Science, pp. 259-282, Springer Verlag, April 1991.
- [6] P. Curzon, Of What Use is a Verified Compiler Specification, Technical Report No.274, Computer Laboratory, University of Cambridge, November 1992.
- [7] M. J. C. Gordon, Mechanizing Programming Logics in Higher Order Logic, in G. Birtwistle and P.A. Subrahmanyam, Eds., Current Trends in Hardware Verification and Automated Theorem Proving, Springer-Verlag, New York, 1989.
- [8] M. J. C. Gordon and T. F. Melham, Introduction to HOL: A theorem proving environment for higher order logic, Cambridge University Press, Cambridge, 1993.
- [9] J.J. Joyce, Totally Verified Systems: Linking verified software to verified hardware. In M. Leeser and G. Brown, Eds., Specification, Verification and synthesis: Mathematical Aspects, Springer-Verlag, 1989.
- [10] Z. Manna and A. Pnueli, Verification of Concurrent Programs: A Temporal Proof System, Proc. of the Fourth School of Advanced Programming, Amsterdam, 1982.
- [11] L. C. Paulson, Logic and Computation: Interactive Proof with Cambridge LCF, Cambridge University Press, Cambridge, New York, 1987.
- [12] V. Yodaiken and K. Ramamritham, Verification of a Reliable Net Protocol, Proc. of the Second International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, number 571 in Lecture Notes in Computer Science, pp. 193-215, Springer Verlag, January 1992.
- [13] C. Zhang, R. Shaw, R. Olsson, K. Levitt, M. Archer, M. Heckman, and G. Benson, Mechanizing a Programming Logic for the Concurrent Programming Language microSR in HOL, in J.J. Gordon and C.H. Seger, Eds., Higher Order Logic Theorem Proving and Its Applications, The 6th International Workshop, HUG'93, number 780 in Lecture Notes in Computer Science, pp31-44, Springer-Verlag, March 1994.
- [14] MACH 3 Kernel Principles, Open Software Foundation and Carnegie Mellon University, May 1991.
- [15] USA INFOSEC Research and Technology, Synergy: A Distributed, Microkernel-based Security Architecture, November 1993.

Making secure dependencies over a LAN architecture for security needs

Bruno d'AUSBOURG

CERTIONERA

Département d'Etudes et de Recherches en Informatique

2, Avenue E. Belin - B.P. 4025

31055 Toulouse - Cedex- FRANCE

email: ausbourg@tls-cs.cert.fr

Abstract

It was recently argued that the presence of covert channels should no longer be taken for granted in multilevel secure systems. To date, multilevel security seems to have been an ideal to approach and not a requirement to meet. The question is: is it possible to design a practical multilevel system offering full security? Based on what architecture? The approach described in this paper reflects some results of a research project which suggests some ideas to answer this question. We have chosen the distributed architecture of a secure LAN as an application framework. In particular we show how controls exerted on dependencies permit to control exhaustively the elementary flows of information. The enforced rules govern both the observation and the handling of data over the whole system. They are achieved by means of some hardware mechanisms which submit the access of hosts to the medium to a secure medium access control protocol. We evaluate how secure dependencies used to ensure confidentiality in such an architecture may also be used to answer some other needs of critical applications with regard to other attributes of security as integrity and availability.

Keywords

Secure dependencies, causality, multilevel security, distributed systems, network protocols

1 Introduction

Many systems have been built in order to protect confidentiality of data and processes. This is generally done by offering multilevel architectures of machines and networks. These architectures tolerate the existence of covert channels, because standards consider that covert channels are inevitable. Proctor and Neumann in [7] argued that the presence of covert channels should no longer be taken for granted in multilevel secure systems. Indeed, applications should not tolerate any compromise of multilevel security, not even through covert channels of low bandwidths. They argued also that systems with multilevel processors seem to be either impractical or insecure. They suggest to redirect research and development efforts towards developing multilevel disk drives and multilevel network interface units for use with only single level processors in building multilevel distributed systems.

The approach described in this paper reflects some results of a research project¹ which suggests some ideas in the same direction. This project aims at building a system architecture (machine and LAN) that offers a high degree of protection both for storage, processing and communication of user data. By high degree of protection we mean a protection that is based on an exhaustive control of information flows, including timing flows, and ensuring there is no place for covert channels. We have chosen the distributed architecture of a secure LAN as an application framework. In particular we show how controls exerted on dependencies permit to control exhaustively the elementary flows of information. The enforced rules govern both the observation and the handling of data over the whole system. They are achieved by means of some hardware mechanisms which submit the access of hosts to the medium to a secure medium access control protocol. We evaluate how secure dependencies used to ensure confidentiality in such an architecture may also be used to answer some other needs of critical applications with regard to other attributes of security as integrity and availability.

1. This project was supported by the French DGA/DEI/STEL.

2. Causal Dependencies and Security

2.1 Causal dependencies

A system may be described as a set of points (o, t) . A point references an object o at a time or date t . This introduction of time is necessary because time can be observed in the system. Through durations of operations for example. So, one can act on the value of the object o , at the instant t , or one can act on the instant t at which the object o is given a particular value. In the first case, the object o can be used to transmit some information if any semantics can be assigned to its value and a storage channel is involved here; in the second case, time is used and therefore, a timing channel is involved if any semantics can be assigned to the observed instant values.

Some of these points are *input* points, others are *output* points, and the last ones are *internal* points. These points evolve with time and this evolving is due to the elementary transitions made by the system. An elementary transition can modify a point: then, at instant t , it sets a new value v for the object o of the point. This instant t and the new value v functionally depend on previous points. This *functional dependency* on *previous* points is named *causal* dependency. The causal dependency of $y = (o, t)$ on $x = (o', t')$ with $t' < t$ is denoted by $x \rightarrow y$. Informally, by y *causally depends on* x we mean that the point x is used to generate the point y . We denote \rightarrow^* the transitive closure of the relation \rightarrow .

These causal dependencies make up the structure of information flows inside the system. If a subject s has some knowledge about the internal functioning of the system, then he is able to know the internal scheme of causal dependencies. So, by observing any output point x_o , he is able to infer any information on points $x / x \rightarrow^* x_o$. In particular x may be an input points x_i which contain some input data of the system. Conversely, by altering an input point x_i , s can alter any point $x / x_i \rightarrow^* x$ and in particular an output point $x = x_o$.

2.2 Security

Let A_s that contains the points on which a user, or more generally a subject s in the system, is able to make an action: observation or alteration. The set R_s contains the points on which the subject s may act (has the right to observe or to alter) in the system, in accordance with the security policy. So, the system is secure if a subject s can act on the only objects he has the right to act:

$$A_s \subseteq R_s \quad (1)$$

When the security policy which is used to define the rights of subjects is the multilevel security policy, a classification level $l(x)$ is assigned to points x and a clearance level $l(s)$ is assigned to subjects s . A convention on levels may be chosen: a level l is a pair (l_c, l_i) where l_c is a level of confidentiality and l_i a level of integrity. When integrity is addressed, value of levels is generally also increasing with the integrity expected on points. The comparison rule on levels may be defined as:

$$(l_1 \leq l_2) \Leftrightarrow (l_{c1}, l_{i1}) \leq (l_{c2}, l_{i2}) \Leftrightarrow (l_{c1} \leq l_{c2}) \wedge (l_{i1} \geq l_{i2})$$

So, the set R_s may be defined quite naturally by $R_s = \{x / l(x) \leq l(s)\}$

When confidentiality is the addressed property, observation is controlled and R_s contains the points that may be observed by the subject s : these are the only points whose classification level is dominated by $l(s)$. When integrity is the addressed property, alteration is controlled and R_s contains the points that may be altered by the subject s : these are the only points whose integrity level is dominated by $l(s)$.

2.3 Security conditions

It is shown in [5] that two conditions are *sufficient* to guarantee the security defined by (1). A first interface rule expresses conditions on the classification level of an interface (input or output) point x_i and the clearance level of the subject s who can observe or alter this point:

$$\forall s, x_o \in A_s \Rightarrow l(s) \geq l(x_i) \quad (2)$$

The second condition requires a monotonic increasing (in the sense of the comparison rule) of levels over causal dependencies.

$$\forall x, \forall y, x \rightarrow y \Rightarrow l(x) \leq l(y) \quad (3)$$

With regard to confidentiality, the both rules (2) and (3) ensure that any subject s who has the right to observe an output point x_o is allowed to observe any information on previous points $x / x \rightarrow x_o$. So the observation of x_o will give to s only information he has the right to observe. With regard to integrity, the both rules (2) and (3) ensure that any subject s who has the right to alter an input point x_i is allowed to alter any point $x / x_i \rightarrow x$. So the alteration of x_i by s will have an impact only on points that s has the right to alter.

These rules define *secure* dependencies. The condition (3) is interesting because it gives the semantics of an internal control which can be exerted on each system transition when a relation of causal dependency is involved. It enforces the exhaustive control of information flows. This control of information flows (including its temporal aspects) is achieved by making sure each transition and each elementary transfer of information from input points until system points which can be observed directly by a user. All information channels are involved (storage and timing) and it exists no potential covert channel.

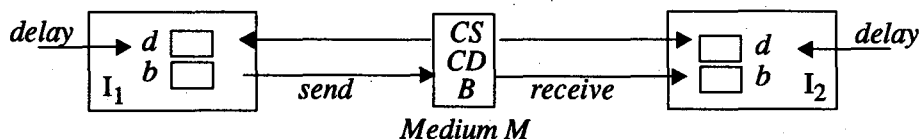
3 Interpretation

These rules may be instantiated by making an interpretation of the model in the context of one of the various abstract layers of a real system. The choice has been made to perform this interpretation in the hardware layer.

3.1 Security conditions in interface units connecting hosts to a LAN

Network interface units U connecting stations to a communication medium constitute the system architecture as described by Figure 1. These units access the medium according to the CSMA/CD Medium Access Control protocol, as defined by IEEE 802.3. We denote by *Medium M* the protocol data units managed by the Physical Layer. In particular, this layer offers two elementary signals (*Carrier Sense, Collision Detection*) and B which contains the bit value carried by M .

Figure 1 The system architecture



The active entities, which are the *subjects* inside this hardware layer, are the network interface units U . These units have one input $delay$ value, that is chosen externally as a uniformly distributed random value in a finite range. They can be represented by two data cells: the bit value b it has to deposit on M or it has sampled from M and d that contains a delay value to spend before transmitting.

In the same way, the *objects* are the internal cells b and $delay$ of U and the communication medium M (including CS, CD and B). A level is assigned to all objects and subjects. The cells b and $delay$ in U have a level $l(U)$ and all objects in M share the same level $l(M)$. The elementary transitions include the elementary $send$ and $receive$ operations made by U between its own cells and M .

The $receive$ operation, as expressed in the CSMA/CD protocol, consists in permanently listening to signals CS and to the bit value B carried by M . This operation produces a new value for and the following dependencies are involved:

$$\{CS, B\} \rightarrow b$$

Condition-receive. In this case, the rule (3) applied to the $receive$ operation gives:

$$l(M) \leq l(U)$$

The *send* operation is less simple. Firstly, the decision by U to deposit a bit value upon M is taken by listening to M and watching at signals CS and CD . The transmission of the b value may be delayed according to the delay value stored in d when CD indicates that a collision occurred. When transmitting the bit b , a new value is assigned to the M components. So

$$\begin{aligned} \{CS, CD\} \cup \{delay\} &\rightarrow \{d\} \\ \{b, d\} &\rightarrow \{CS, CD, B\} \end{aligned}$$

Condition-send. The rule (3) applied to these dependencies gives

$$l(M) \leq l(U) \leq l(M) \Rightarrow l(M) = l(U)$$

3.2 Management of level objects

Levels are themselves objects in the system. So they are also submitted to the control of dependencies. A classification level is assigned to them: we have chosen to give the value "Low" to the level of a level object. Then, the fact that an information is secret is not itself a secret. It is *not a doctrine*, but only a *work assumption* that we made in order to simplify.

Being submitted to the control of dependencies, the rule (3) must be applied to levels and then, given a level l_i :

$$x \rightarrow l_i \Rightarrow l(x) \leq l(l_i) \Rightarrow l(x) = Low$$

In other words, the value of a level and the instant at which this level gets a given value only depend on low level information. This condition is sometimes difficult to enforce, for example, when the value of a level decreases from a *High* to a *Low* value. This change of the level value must have been planned and declared at *Low* level.

In our system architecture, the value of the level of M , and time at which this level takes a given value must be generated from *Low* level points. Then, the value of the level of the medium and the time spent to this level are stated at *Low* level. Therefore, the use of M is time sliced between levels. And slices are declared or computed at *Low* level. A *High* process *never* acts on the value of a level (by maintaining it or by changing it).

Similarly, the level of U must be declared at low level. And the time spent by U at this given level is also declared in advance at *Low* level. So at the beginning, U is at *Low* level. If a user wants to use the host and U at a level *High*, this user (and not a process running on the untrusted host) must firstly declare at *Low* level (not *High*) that he requires to use the unit U at level *High* during time t , in order to achieve communications at level *High*.

3.3 Security SubSystem: S^3

Because they are quite simple, the controls can be enforced by a subset of hardware features which are driven by a subset of software. These two subsets constitute the *Security SubSystem* or S^3 of the whole system. This S^3 is also built inside a multilevel machine.

3.4 Interpretation inside a Machine for Multilevel Security: M^2S

M^2S is a machine built upon these principles and was fully described in [4]. M^2S combines a processor with an address space A . The processor addresses the space A when executing elementary transfers to external devices as memory, registers of controllers.

The elementary objects that can be observed by a user comprise processor registers r and cells a of the address space. Levels are assigned to these objects. The level assigned to the processor registers determines the *current level* cl of the whole system. A value is assigned to cl by SSS in accordance with rules of time slicing. Assigning levels to cells of the address space divides it in different partitions. Each partition of this address space may be reached by the processor according to the value of its current level cl , the requested access mode and rules of flow control.

The state of the system is reflected by the state of processor registers and the state of buses (address, data and control). Inside the system, internal information flows are caused by elementary

transfers between the processor and the address space: transfers of data or transfers of interrupt signals. The S^3 is in charge of controlling these information flows. It does it by making use of specific hardware components which are under control of a Security Processor (PS). This PS uses its own resources in order to store and to manage security data.

Figure 2 Elementary transfer controls inside M^2S

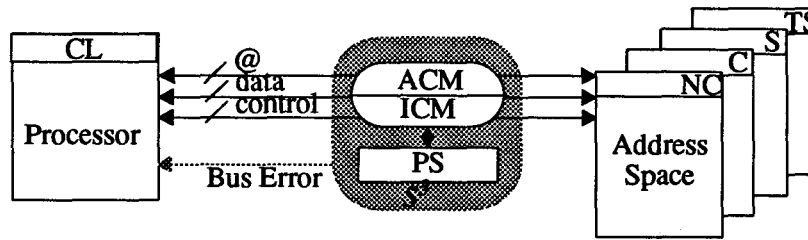


Figure 2 describes how the S^3 controls the information flows inside the system. In fact, the S^3 inspects the state of buses in real time. It determines which states are allowed in accordance with security data and with rules of flow controls. If an illicit state is reached during an elementary cycle, this cycle is interrupted by PS and a Bus Error signal is sent to the Processor.

The Access Control Module (ACM) controls the first kind of transfers inside the system. At current level cl of the processor, a read cycle to an address a of level l_a involves a dependence ($a \rightarrow r$) where r is a register of the processor. The rule (3) implies $l_a \leq l(r) = cl$. In the case of a write cycle, the dependence ($r \rightarrow a$) is involved: so $l(r) = cl \leq l_a$. This module comprises an additional component which is in charge of controlling transfer operations that use a more complex addressing mode. In particular, when an access to a disk data block is involved, the processor uses the data bus in order to transfer some addressing information to the disk controller: cylinder, track number, sector number...

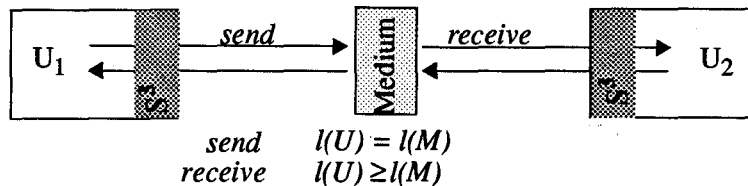
The Interrupt Control Module (ICM) controls the second kind of transfers. It filters interrupt signals emitted by peripheral devices which are located somewhere in the address space. If the signal sender is an object whose level is l_o , the interrupt signal is also an object whose assigned level is $l_i = l_o$. This signal causes a dependence ($o \rightarrow p$). ICM transmits this signal to the processor when $cl = l(p) \geq l_i$. In any other case, the signal is suspended until the condition becomes valid. In practice, in order to handle them more easily, interrupt signals are received by the processor when $cl = l_i$.

4 Distributed S^3 over a LAN

4.1 Enforcement of the security conditions in local S^3

The security condition expressed in (3) can be applied also to the operations of sending and receiving bits which are done by the network interface units. A local S^3 is in charge of enforcing these controls and to regulate the access of these interface units to the communication medium.

Figure 3 Rules to access the Bus in a network interface unit

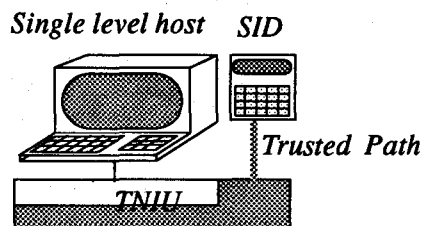


This local S^3 keeps values of levels for the interface unit and for the communication bus. It grants or denies to the interface unit the access right to the bus according to those values of levels. In fact, it can intervene by hardware on elementary operations exerted in order to deposit or sample information on the medium. So, for the interface unit, the ability to send or receive at any instant t is given by its own level and the level of the bus. An interface unit equipped with its local S^3 constitutes a Trusted Network Interface Unit or TNIU.

4.2 Trusted paths to local S³

There is a need for building a trusted path between users and local S³ of the network interface unit. The mechanism of a Secure Interface Device (SID) is used and is shown in Figure 4.

Figure 4 Trusted Path



A quite simple dialogue between users and local S³ permits to declare the value of the current level of the connected host for the next session and the required duration for this session; this fixes the level of the interface unit and the time needed for exchanges at this level and this permits to initialize then the local S³ functioning.

4.3 Security Subnetwork

Exchanges between hosts running at various current levels may occur only if the level of the bus can change. In fact, this value must be time sliced in accordance with rules defines in paragraph 3.2: this slicing is based on level reservations which are produced and emitted at low level. Then, two conditions must be satisfied. Firstly, the value assigned to the level of the bust must be known by every local S³. Secondly, the time slicing of this value must be enforced in a synchronous way over the LAN.

Satisfying the first condition requires a communication subnetwork between all the local S³. In fact, in this case, this subnetwork uses the same bus of communication as hosts. The local S³ which are interconnected by this way constitute the *security subnetwork* of the system. This security subnetwork is used to exchange security data between local S³.

A *centralized security station* (or CSS) manages the data of security for the network. In particular, it manages levels which are assigned to interface units and to the communication bus according to reservations made by hosts and users through the SID and emitted to CSS by local S³. It broadcasts also these data to all the local S³ over the security subnetwork.

Satisfying the second condition requires the existence of a protocol in charge of regulating the exchanges of security data. It is also in charge of ensuring that the time slicing of the bus level is known by all the local S³ in a synchronous manner. So, the rules which are used to access the bus in order to exchange security data are not the same as the rules used by hosts in order to exchange user data. These rules constitute the *Security Medium Access Control (SMAC)* protocol.

5. SMAC protocol and multilevel LAN

5.1 SMAC Protocol

It enforces time slicing for the level of the bus according to reservations made to the CSS. It manages also the exchange of security data under the authority of the CSS. These data include particularly reservation data emitted from local S³ and level settings for the bus which are emitted from the CSS. In few words, the SMAC protocol is reservation based.

It manages two functioning modes for the interface unit: a *user* mode and a *security* mode. In the security mode, only local S³ can use the bus to exchange security data with CSS. In user mode, operations to send and receive user data can be performed by the interface units according to values of their own level and of the level of the bus. The CSS computes time slices for sessions of exchanges in user mode which correspond to various values assigned to the level of the bus. These values are set in accordance with reservations previously received. At the end of a slice, the interface unit always returns to the security mode. In security mode, the CSS may ask to local S³ if reservations are pending.

If yes, local S^3 may answer by giving the content of their pending reservations. The protocol for this dialogue is a synchronous one. The CSS fixes a transmission slot for each local S^3 to answer and each local S^3 may answer during its reserved slot. The CSS broadcasts then a new value for the level of the bus and a new session in user mode is started. In user mode, a Medium Access Control (MAC) protocol arbitrates the access to the bus between units which are allowed to access it: this protocol is CSMA/CD is in our case.

The SMAC protocol is similar to protocols used in the real time world where requirements on the amount of delay between the time a packet is ready and the time it is received at destination are stringent. In these protocols, some sources must reserve transmission slots before they can begin transmission [8].

5.2 DS^3 and multilevel LAN

The CSS, the local S^3 and the Bus which is accessed in accordance with the rules of the SMAC protocol constitute the *Distributed S^3* of the LAN (or DS^3). This architecture may be built upon a yet existing Ethernet LAN. The DS^3 and the local S^3 cooperate in enforcing the control of information flows in the more concrete layer of the system: the hardware layer. The aimed controls tend to master the involved causal dependencies by verifying their accordance to the rules of multilevel security.

A multilevel station, built above the same principles (more details in [4]) may be added to ensure a secure sharing of data between levels. Because this multilevel station is able to manage multilevel data structures and processes, it permits to single level stations to access data through levels in a quite secure manner.

The global architecture of such a system constitutes a secure LAN which is said to have a multilevel functioning mode. Such an architecture and its multilevel functioning mode verify the required security property: all the information flows, including timing flows, are controlled exhaustively. There is no way to build any covert channel.

It is obvious that this architecture is insufficient if the communication bus is vulnerable: that is not the addressed problem in this paper. Cryptographic techniques may be added to preserve the confidentiality and integrity of messages transmitted over the network. These techniques may rely on cryptographic devices and functions which can be driven by the Distributed S^3 (local S^3 and CSS). They can be viewed as an external protection layer, by opposite to the internal protection layer described here.

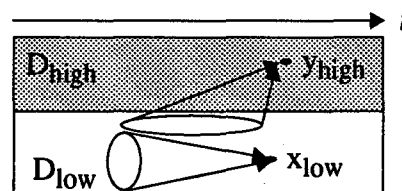
6 Discussion

Such an architecture enforces the rules of multilevel security. The DS^3 aim at controlling internal information flows which are involved when communications are achieved over the medium by ensuring that the involved causal dependencies are secure. This control of information flows may be used in order to enforce confidentiality, integrity and availability properties.

6.1 With regard to confidentiality

Let x_{low} and y_{high} which are two points which belong to two different domains D_{low} and D_{high} in the system. These domains may be defined, when multilevel security is the applied security policy, by $D_l = \{x / l_c(x) = l\}$ if $l_c(x)$ is the confidentiality level of x , with $\cap D_l = \emptyset$.

Figure 5 Graphical illustration of confidentiality properties

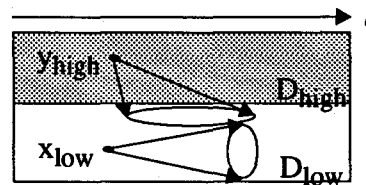


The security condition (3) and its enforcement in SMAC ensures that the observation of any point in D_{low} will reveal no information about points in D_{high} . But points of D_{high} may be built from points of D_{low} . The only allowed flows of information are from *low* to *high*. It is a classical result in confidentiality. In this case, this result is applied in the context of exchanges which are achieved between hosts at different levels.

6.2 With regard to integrity

Let x_{low} and y_{high} which two points that belong to two different domains of integrity D_{low} and D_{high} in the system. These domains may be defined, when multilevel security is the applied security policy, by $D_l = \{x / l_i(x) = l\}$ if $l_i(x)$ is the integrity level of x . So D_{low} denotes a domain of low integrity and D_{high} denotes a domain of high integrity.

Figure 6 Graphical illustration of integrity properties



The security condition (3) and its enforcement in SMAC ensures that the alteration of any point in D_{low} will alter no information about points in D_{high} . So the only allowed flows of information are from *high* to *low*. This is in accordance with classical results as expressed by Biba [2] for example.

By defining integrity domains and by controlling flows between these domains according to the previous rules, we ensure there is no way, at a low integrity domain, to use any input covert channel in order to insert corrupted instructions or data in a high integrity domain.

These results may be applied to isolate and minimize functions which are vital to run a critical process inside a global environment (see the discussion in [6]). Criticality levels may be defined; they reflect the degree of criticality of functions or data to the system objective. So a *High* critical domain is fully protected from eventually malicious operations exerted from a *Low* critical domain. This scheme is interesting in a security point of view, but also for cost considerations. Indeed, it permits to minimize the *High* critical domain by including in it the only really critical functions and data. Techniques used during the development of such a system and during its running in order to ensure dependability properties may be reduced by limiting them to the only critical domain.

6.3 With regard to availability

A particular case of the integrity property which was previously described offers some kind of availability. Indeed, the SMAC protocol which is used to share the communication medium of the multilevel LAN tends to separate domains of integrity/criticality and to regulate flows between these domains according to multilevel rules.

In particular, the time slicing exerted on the level of the medium coupled with the ability assigned to the interface units of sending or receiving according to time slices make impossible for an interface at a *Low* integrity level to disrupt the use of the communication medium by interfaces at a *High* level of integrity. When communications occur at a given level, there is no way for interface units at an other level to get any send access to the medium.

So the availability of services inside the domain of *High* integrity can not be countered by malicious processes at a *Low* level of integrity or by a crash or a bad functioning occurring on an host at a lower level of integrity.

So, some mechanisms may be employed to ensure high availability inside the high integrity domain itself. But their use is limited inside this domain only, and the availability property is not put in danger by lower integrity levels, thanks to the separation enforced by the DS³ and the SMAC protocol.

7 Conclusion

Techniques and mechanisms suggested here were firstly designed and developed in order to protect the confidentiality of data, processes and communications over a LAN. This high protection is based on a control of dependencies inside the system, ensuring that all dependencies are in accordance with multilevel security rules. In fact, these controls enforce an exhaustive control of information flows. They rely upon a distributed security subsystem composed of a particularly restricted subset of hardware mechanisms: they are in charge of ensuring that accesses of interface units to the medium are done in accordance with multilevel rules. This leads to share the medium in a particular way which defines a secure medium access control (or SMAC) protocol. This protocol may be viewed as an extension of an existing MAC protocol, as CSMA/CD.

The logical separation achieved by means of this protocol may be used to separate integrity levels. In particular, the extremely strong control of information flows which is enforced can isolate some domain where a high level of integrity may be needed drastically. This domain is then protected from other domains of low integrity that can not corrupt its behaviour: in particular they can not enforce any communication channel to send malicious data or pieces of code. Such levels of integrity can be used in critical applications to protect some vital functions. As a particular case of the application of control of dependencies to integrity, some needs in availability may be answered also. The separation between high integrity and low integrity domains ensure that any (malicious or not) failure in a low integrity domain will not disrupt the good functioning inside a high integrity domain.

This whole security protects efficiently all the information that needs to be protected, and only this information. We feel that this approach is well adapted to the real world, where in fact, few informations and functions necessitate to be protected. So, such a system does not penalize the use and processing of most of the data which belong to an unprotected domain. Rather, it makes lighter the amount of protected processing by reserving it to the only data which necessitate it.

8 References

- [1] P. Bieber, F. Cuppens
A logical view of secure dependencies. In *Journal of Computer Security*, Vol. 1, Nr. 1, IOS Press, 1992
- [2] K. J. Biba
Integrity Considerations for Secure Computer Systems, Technical Report ESD-TR-76-372, ESD/AFSC, Hanscom AFB, Bedford, Mass., 1977. Also MITRE MTR-3153.
- [3] D. E. Bell and L. J. Padula
Secure Computer Systems: Unified Exposition and Multics Interpretation, MTR-2997, MITRE Corporation, Bedford, Mass. (1975).
- [4] B. d'Ausbourg and J.H. Llaeus
M²S: A machine for multilevel security, *European Symposium on Research in Computer Security, ESORICS92*, Toulouse, France, 1992
- [5] G.Eizenberg.
Mandatory policy: secure system model. In AFCET,editor, *European Workshop on Computer Security*, Paris,1989.
- [6] H.L Johnson et al.
Integrity and Assurance of service Protection in a large, multipurpose, critical System, *In Proceedings of the 15th National Computer Security Conference, Baltimore, MD, October 1992*
- [7] N. E. Proctor and P. G. Neumann
Architectural implications of covert channels, *In Proceedings of the 15th National Computer Security Conference, Baltimore, MD, October 1992*
- [8] R. Yavatkar, P. Pai and R. Finkel
A reservation based CSMA Protocol for Integrated Manufacturing networks, *Tecn. Rep. 216-92, Department of Comp. Sc., Univeristy of Kentucky, Lexington, KY*

AUTOMATIC GENERATION OF HIGH ASSURANCE SECURITY GUARD FILTERS *

Vipin Swarup
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730-1420
(swarup@mitre.org)

Abstract

This paper presents a generic architecture for security filters and a methodology for developing a wide variety of high assurance security filters. We define a security filter specification language (called *Felt*) which can specify the behavior of a wide range of security filters. We describe a *Felt* compiler which automatically translates *Felt* specifications to security filter programs. These filter programs are written in C and can be compiled and run on a wide range of computers. An important aspect of this work is its emphasis on achieving high assurance without sacrificing performance or cost. Towards this end, we have provided *Felt* with a formal semantics in the denotational style. We have also formally verified non-trivial algorithms used in our *Felt* compiler.

1 Introduction

A *security guard* mediates data transfer across security boundaries. It may be a separate device, program, or manual controls. A guard is typically used to control data transfer between systems that are at different security levels of protection and have different user profiles. Guards are also used in single systems (e.g., databases) to downgrade information.

A *security filter* provides the essential filtering services of a security guard. It decides whether to allow data to cross the security boundary mediated by the security guard. In certain cases, it transforms data (e.g., sanitization, reformatting, etc.) and permits the transformed data to cross the boundary. It maintains logs of the data it filters and the actions taken on the data (e.g., the sanitization

performed, the reasons for preventing data from crossing the boundary, etc.)

In this paper, we make the following contributions:

1. We present a generic security filter architecture that can be used as the basis for comparing security filters or developing new security filters. The architecture includes message formats, filtering constraints, message transformations, and logging actions.
2. We present a new methodology for developing a wide variety of high assurance security filters. The methodology is based on a *security filter specification language* (called *Felt*) which can specify the behavior of a wide range of security filters. *Felt* specifications can be compiled to security filter programs and run on a wide range of computers. We have provided *Felt* with a formal denotational semantics which precisely specifies the meanings of *Felt* specifications. We have formally verified the non-trivial algorithms used in our *Felt* compiler, thus providing high assurance that generated security filter programs have the same meanings as their specifications.

The advantages of our approach relative to existing security filters (such as filters in the WWMCCS guard [2], USAFE guard [3], etc.) are as follows.

Extensive Filtering Capabilities: *Felt* is a very expressive specification language. For example, constraints involving relations between the values of several fields can be specified simply and clearly. A wide variety of security filters can be specified in *Felt*, and *Felt* can evolve to accommodate new requirements on security filters.

In contrast, existing security filters embed much of their functionality within custom-

*Work supported by Rome Laboratory, Electronic Systems Command, United States Air Force under contract F19628-93-C-0001.

developed program code. This makes it difficult and expensive to develop and maintain security filters with advanced features such as message sanitization and inter-field constraints. Thus, these filters typically support only some of the filtering capabilities supported by *Felt*.

Widely Configurable: Any specifiable aspect of the behavior of a security filter can be altered by modifying the high-level *Felt* specification of the filter and recompiling the specification. This includes message formats, constraints, message transformations, and logging actions. In contrast, existing security filters only permit some of their filtering capabilities to be reconfigured (via predefined context tables).

Good Performance: A security filter program is obtained by compiling a *Felt* specification for the filter. *Felt* contains a small number of basic primitives, and a *Felt* compiler can be designed to compile these primitives to very efficient code. For example, our compiler compiles table lookups and dirty-word searches into language recognizers based on efficient deterministic finite state automata (DFA).

Low Cost: The development cost of *Felt* and a verified compiler for it is a one-time cost (approximately 6 man months, thus far). Thereafter, the only cost involved in creating and modifying a security filter is the cost of creating and modifying a filter specification. Since *Felt* is a special-purpose language designed for the specification of security filters, it is a simple, intuitive, high-level language. Further, our examples show that filter specifications are approximately a tenth the size of corresponding filter programs. This should make it easier, less error-prone, and cheaper to develop and modify specifications as opposed to developing and modifying program code. These potential benefits should increase considerably with the development of a good user interface.

High Assurance: The behavior of a filter is specified precisely using a specification language (*Felt*) that has a formal semantics—there is no ambiguity about what the specifications mean. Further, nontrivial algorithms of *Felt*'s compiler have been formally verified. In contrast, the behavior of existing security filters is represented within large computer programs (written in C or Ada, for example). This

is quite error-prone, and little assurance exists that the computer programs will filter all messages correctly.

The remainder of this paper is organized as follows. Section 2 presents a generic architecture of security filters. Section 3 describes the syntax and semantics of a security filter specification language (*Felt*). Section 4 describes a *Felt* compiler that translates *Felt* specifications to security filter programs. Section 5 discusses issues related to the assurance of the resulting security filters. Section 6 concludes this paper with a discussion of possible extensions to *Felt*.

2 A Security Filter Architecture

Figure 1 depicts a generic architecture of security filters. In this architecture, security filters have four primary aspects: message parsing, constraint checking, transformations, and actions. A security filter accepts as input a stream of bytes. The *message parser* partitions the byte stream into a sequence of *messages*. Each message is examined by the *constraint checker* to see if it meets predefined criteria. The constraint checker can invoke the *message transformer* to transform messages during this process. The constraint checker and message transformer can log their activities using *actions*. The constraint checker can also use actions to write (possibly transformed) messages to "accept" and "reject" output streams depending on whether or not the messages meet the predefined criteria.

A *message* consists of a message header followed by a sequence of bytes. Messages are grouped into message categories. Message formats, constraints, transformations, and actions are associated with categories so that all messages in a category are filtered similarly. Message categories can be either fixed-format or variable-format categories. A fixed-format category contains messages of a fixed length. Further, the messages are partitioned into fixed-length fields. The contents of certain fields can be restricted to certain types of data. For example, a field can be restricted to contain numeric data only. A variable-format category contains messages whose length and format are only partially predetermined, i.e., messages in the same variable-format category can have different lengths and formats.

The message header of a message uniquely identifies its message category. In the case of fixed-format categories, this automatically identifies the length

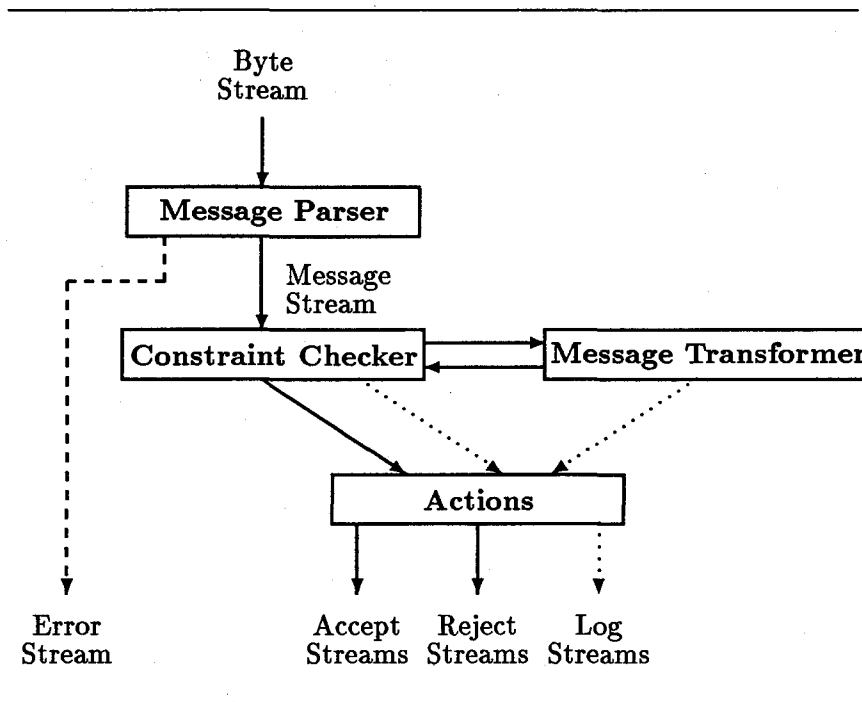


Figure 1: A security filter architecture.

and format of the message. In the case of variable-format categories, the header includes the length of the message. Format information can be encoded within the message if desired. The message parser uses message headers to partition the input byte stream into a sequence of messages.

Constraints can be either integrity constraints (that check whether messages contain legal data) or confidentiality constraints (that check whether messages are free of any restricted data). They can constrain data to contain certain values, e.g., a field must contain numeric characters. They can constrain data to be free of certain values, e.g., a field must be free of restricted words (“dirty words”). They can constrain data to bear a certain relationship with each other, e.g., the date in a field must be less than or equal to the date in another field.

Transformations are functions that transform messages to new messages. Two important classes of transformations are sanitizing transformations and reformatting transformations. Sanitizing transformations sanitize messages by replacing the contents of fields with less sensitive data, e.g., blanks or the contents of other less sensitive fields. Reformatting transformations modify the formats of messages, e.g., elimination of fields, rearranging of fields, etc. Transformations can be either unconditional transformations (in which case they are

applied to all messages of their category) or conditional transformations (in which case they are only applied to messages of their category that meet a constraint). Note that since transformations alter the contents of fields, they can invalidate inter-field constraints that held prior to the transformation.

Actions are procedures that cause the filter to change its state. Two important classes of actions are filtering actions and logging actions. Filtering actions write messages to output streams. Such actions are invoked by the constraint checker to partition the input message stream into several “accept” and “reject” streams. For example, messages that do not meet the predefined constraints can be written to multiple “reject” streams depending on their importance. This can, for instance, permit a human reviewer to scan important messages first. Logging actions cause data to be written to log files. For example, such actions can be used to log the reasons for messages to be sent to “reject” streams. They can also be used to log the sanitization operations (and other transformations) performed on messages.

This architecture assumes that the input and output streams of a filter are established by code external to the filter. It also assumes that the external code determines the message category and length of each data set, attaches the appropriate header to

the data set, then sends it along the input stream as a sequence of bytes. Similarly, the external code processes the data on the output streams. This external code depends on the operating environment of the security filter. This includes both the operating system of the computer as well as the security guard within which the filter is embedded.

3 Felt: A Security Filter Specification Language

*Felt*¹ is a specification language which can express the behavior of a wide range of security filters. The language is based on the general architecture described in Section 2. Thus a *Felt* specification of a security filter includes, for each message category, the identification characteristics that uniquely identify the message category, the format of messages in the category, the constraints that messages should satisfy in order to be "accepted", rules for transforming messages, and actions to be taken by the security filter at various stages of its execution. In this section, we describe *Felt* informally. The syntax and formal denotational semantics of *Felt* is presented in [10].

A filter specification begins with a sequence of constraint definitions, transformation rule definitions, and action definitions. These definitions assign names to constraint functions, transformation rules, and actions respectively. The names can be used elsewhere in the specification in place of the definitions. Note that definitions can be recursive. These definitions are followed by a sequence of message category definitions. A message category definition specifies the parsing and filtering behavior for all messages in a category. The following subsections describe the components of a message category definition.

3.1 Message Header and Length

Felt assumes that every message begins with a header. A header is specified as a fixed-width string that satisfies a constraint. Note that the headers of each category must be distinct from headers of other categories, and no header should be a prefix of any other header. This implies that every message has a unique valid header, and the header uniquely identifies the category of the message.

Felt also assumes that the length of a message can be uniquely determined from the message's

header. If all messages in a category have the same length, then the length is specified as a constant number; since a header uniquely identifies the category of a message, it uniquely identifies the length of the message. If messages in a category have different lengths, then it is assumed that the length is encoded within the header and hence the length is specified as an explicit function of the header. For example,

```
(define-message tpfdd-non-unit-record
  (header 1 (function (x)
    (or (equal-to-string? x "G")
        (equal-to-string? x "J"))))
  (length 18)
  ...
```

specifies a message category that is called *tpfdd-non-unit-record*. Every message in the category is 18 bytes long and has a header of length 1 that is either the string "G" or the string "J". As another example,

```
(define-message tclcf-record
  (header 5 (function (x)
    (equal-to-string?
     (substring x 0 1)
     "F")))
  (length (function (x)
    (string->number
     (substring x 1 5))))
  ...
```

specifies a message category that is called *tclcf-record*. Every message in the category has a header of length 5 whose first byte is the character 'F'. The length of the message is encoded (as a binary number) within the remaining four bytes of the header.

The message header and length specification is adequate to partition an input byte stream into a sequence of messages. However, if the input byte stream contains erroneous data and does not represent an exact sequence of messages, then it is desirable for a security filter to recover from the input errors gracefully and resume normal filtering. Currently, *Felt* does not provide constructs for the user to specify error handling and recovery, but relies on the default error handling capability provided by the *Felt* compiler.

3.2 Message Format

Felt permits messages to be partitioned into fields. Constraints are then applied to these fields. Fields and constraints are associated with message categories; hence every message in a category is partitioned into fields in the same way. Fields in a

¹Pieces of *felt* were used as filters in medieval Germany. The word "filter" derives from this.

category have fixed lengths, except that the last field of a message may be of variable length to accommodate variable length messages in a category. *Felt* does not provide for arbitrary variable format messages. However, *Felt* can specify the message formats of numerous guards, including the WWM-CCS and USAFE guards.

A message category definition includes a sequence of field specifications. Each field specification consists of a name that uniquely identifies the field, a string that describes the field, and the length of the field (in number of bytes). The order of the field specifications is significant and represents the order of the fields within a message. Thus the first field specification in this sequence describes the first field of the message, and so on. The length of the last field may be specified as * to denote a variable length field. This field is assumed to span the rest of the message. For example,

```
(define-message tpfdd-non-unit-record
...
(fields
  (n01 "Record Type" 1)
  (n17 "POD EAD" 4)
  (n18 "POD LAD" 4)
  (n29 "Personnel Requiring Transport" 5)
  (n30to32 "Cargo Category Code" 3)
  (n101 newline 1)
)
...
)
```

specifies that every message in the category called *tpfdd-non-unit-record* has six fields. The first field is called *n01* and spans the first byte of the message, the second field is called *n17* and spans the next four bytes of the message, etc. The strings "Record Type", "POD EAD", etc. are textual descriptions of the field; they provide documentation to the user but play no semantic role. As another example,

```
(define-message tlcf-record
...
(fields
  (t01 "Message Header" 5)
  (t02 "Message Body" *)
)
...
)
```

specifies that every message in the *tlcf-record* category has two fields. The first field is called *t01* and spans the first five bytes of the message. The second field is called *t02* and spans the remainder of the message.

3.3 Constraints

A message category definition includes a constraint specification that specifies the requirements for a message to be "accepted". Messages in the category that meet the constraint are "accepted" by the filter while other messages are "rejected". The specification language for constraints is quite powerful and is described below.

A constraint function is a boolean-valued function that takes one or more strings as argument. If a constraint maps strings to true, we say that the strings *satisfy* the constraint. Otherwise, we say that the strings do not satisfy the constraint. For example, "(function (x) (blank-string? x))" is a constraint that accepts a string *x* as argument and returns true if all characters of *x* are the blank character, and false otherwise. The string " " satisfies the constraint, while the string "xyd" does not. A constraint function can be assigned a name by a constraint definition.

Strings can be either constant strings or substrings of messages (e.g., fields and subfields). Constant strings are enclosed in double quotes (e.g., "Comic"), fields are referenced by their names, and subfields are specified as follows:

(substring *S* *N*₀ *N*₁): The substring of *S* from index *N*₀ through index *N*₁ - 1.

A constraint is a boolean-valued expression. *Felt* provides several primitive constraint expressions which can be used to define other constraints. Let *S*, *S*₀, *S*₁, ..., *S*_{*n*}, *S*^{*} be string specifications, *N* be a number, and *CF* be a constraint function. The following are some of the primitive constraint expressions provided at present:

(numeric-string? *S*): True if all characters of *S* are digits.

(is-of-length? *S* *N*): True if the length of *S* is equal to *N*.

(equal-to-string? *S*₀ *S*₁): True if *S*₀ is equal to *S*₁.

(is-a-word-in-list? *S* (*S*₀ *S*₁ ... *S*_{*n*})): True if *S* is equal to one of the strings *S*₀, *S*₁, ..., *S*_{*n*}.

(free-of-words-in-list? *S* (*S*₀ *S*₁ ... *S*_{*n*})): True if *S* does not embed any of the strings *S*₀, *S*₁, ..., *S*_{*n*}.

Constraint functions applied to strings are also constraint expressions:

(apply-constraint CF S*): True if strings S* satisfy the constraint function CF.

Complex constraint expressions can be built from simpler constraint expressions. Let C_0, C_1, \dots, C_n be constraints. The following constructors are provided to construct complex constraint expressions:

(and $C_1 C_2 \dots C_n$): True if each of the constraints C_1, C_2, \dots, C_n is true.

(or $C_1 C_2 \dots C_n$): True if at least one of the constraints C_1, C_2, \dots, C_n is true.

(not C): True if the constraint C is not true.

(if $C_1 C_2 C_3$): True if either C_1 and C_2 are both true, or if C_1 is false and C_3 is true.

Finally, constraint expressions include boolean-valued operations on numbers, characters, and booleans. *Felt* includes coercion operations to coerce string values to these data types.

3.4 Transformations

Felt provides sanitizing and reformatting transformations. Sanitizing transformations are specified as rules (functions) that transform strings to new strings of the same length. A sanitized message must belong to the same category as the original message. Thus the partitioning of a message into fields is unaffected by such transformations.

Felt provides several primitive sanitizing transformation rules which can be used to define other rules. Let S, S_0, S_1 be string specifications. The following primitive sanitizing transformation rules are provided at present:

(replace-with-string! $S_0 S_1$): Replaces all characters in S_0 with those in S_1 .

(map-character-transformation! F S): Transform S by applying the function F to each character in S.

The function F above can be any function from characters to characters. *Felt* provides some built-in functions such as **lower-case** and **upper-case** which return the lower or upper case of the argument character. Others can be created and named within a transformation definition, e.g.,

```
(define-character-transformation
  blank-string!
  (function (x) # ))
```

Reformatting transformations are specified as rules (functions) that transform messages to new messages. A reformatted message is intended to belong to a specific category that is usually different from the category of the original message. Thus a reformatting rule includes a category name. After such a rule is applied, the (reformatted) message will be subject to the format, constraints, transformations, and actions of the specified category. If the message does not belong to that category, the format check will fail and the message will be rejected.

(reformat-message! S I): Replace the current message with the string S and filter the reformatted message S according to the specification of category I.

Felt includes standard string manipulation primitives to permit the string S to be constructed from the fields of the original message.

Transformation rules can be unconditional (in which case every message gets transformed) or conditional (in which case only messages which meet the condition are transformed). A conditional transformation rule is specified as follows:

(conditional-transformation C T): Perform transformation T if and only if the message satisfies the constraint C.

Finally, a sequence of transformation rules is specified as follows:

(begin $T_0 \dots T_n$): Perform transformations T_0, \dots, T_n in sequence.

3.5 Actions

Actions are specified as commands that modify the global states of security filters. Currently, only input and output streams can be modified by actions in *Felt*.

(write-to-stream! S P): Writes the string S to output port P.

(newline-stream! P): Writes the newline character to output port P.

(peak-stream! S P): Transforms the string S to contain the first characters from input port P. The input stream is left unchanged.

(drop-stream! N P): Drops the first N characters from input port P.

4 A Compiler for *Felt*

We have implemented a prototype *Felt* compiler that automatically translates *Felt* specifications of security filters to compilable (then executable) security filter programs. The *Felt* compiler is written in the programming language Scheme [6] and can be executed using any Scheme implementation [1, 4]. The generated security filter programs are written in the programming language PreScheme [7, 8] and can be compiled to C code using a modified version of the verified PreScheme compiler described in [8]. This C code can be compiled to executable code using any C compiler.

A generated security filter program contains a top-level PreScheme loop that partitions the input stream into a sequence of messages and separates the messages into output streams. It first searches for a category definition such that the first few characters in the input stream form a valid message header for the category. If no such category definition is found, a character is removed from the input stream and appended to an "error" file, and the loop is repeated. Otherwise, the length of the message and hence the message itself can be identified. The transformation rules of the selected category are applied to the message. Then, the constraint of the selected category is applied to the input buffer to compute a boolean value. The message is written to one of the output buffers based on the action rules associated with the constraint. *Felt's* semantics ensures that each message is written to at least one output stream; if no explicit actions are provided in the specification, then messages are written to default "accept" and "reject" streams based on the value computed by applying the constraints. The loop is then repeated (after removing the message from the input stream). The loop terminates when the input stream is empty.

The compiler algorithms are straightforward for the most part. The compiler includes two optimizations that significantly boost the efficiency of the generated code. First, in the generated security filter programs, all strings are represented as pairs of start and end indices into an input buffer. The input buffer is a vector of characters that represents the head of the input stream of the filter. This causes the filters to have storage requirements that are statically determinable and obviates the need for dynamic data. The second involves the translation of the primitive constraint functions `is-a-word-in-list?` and `free-of-words-in-list?`. The compiler implements these constraints using *language recognizers*

based on *finite automaton*. This translation has algorithmically significant content and is described in the following subsection.

4.1 Language Recognizers

A *recognizer* for a language is a program that takes as input a string x and answers "yes" if x is a sentence of the language and "no" otherwise. For example, consider the language consisting of all strings that embed one or more of the strings "NATIONAL" and "USA". A recognizer for this language would answer "yes" for the strings "NATIONAL", "FOREIGN NATIONAL", "CRUSADE" (since this embeds "USA"), etc., and "no" for the strings "RATIONAL", "FSA", etc. Language recognizers can be used in integrity and releasability checks to determine whether a string belongs to a set of strings. Another direct application of language recognizers is in dirty-word searches, where the goal is to determine whether a string contains one of a fixed set of "dirty" words.

In *Felt*, a language is specified as a list of *words*, where a word is a sequence of constant characters. No wildcards, character ranges, etc. are permitted. The notation can easily be extended to the regular expression notation (which permits wildcards, character ranges, etc.). However, note that the use of wildcards can result in recognizers with enormous tables of data. The size of these tables can be reduced at the expense of increased execution time—this may or may not be acceptable depending on performance requirements. Algorithms for recognizers are well-known and in common use.

The algorithm for translating a language specification into a recognizer for the specified language has the following steps:

1. Translate the language specification into a *nondeterministic finite automaton* (NFA) that recognizes the specified language;
2. Translate the NFA to an equivalent *deterministic finite automaton* (DFA);
3. Translate the DFA to an equivalent minimum-state DFA that is unique up to state names;
4. Translate the DFA to an equivalent DFA that differs from the former only in the names of states;
5. Translate the DFA to a PreScheme program that is a recognizer for the specified language.

Following [5], we have formally verified that the algorithm does indeed translate a language specification into a recognizer for the specified language. The algorithm and proof of correctness for each of these steps is detailed in [10].

5 Assurance

We have provided *Felt* with a formal semantics in the denotational style [9]. The semantics provides a precise mathematical meaning to specifications written in *Felt*. The denotational semantics is presented in [10] and is having a significant impact on our work. It serves as a language description tool and is influencing the design of *Felt* as a language with a simple, intuitive semantics. In fact, the language described in this paper bears little resemblance to the language that predates the formal semantics. The semantics also serves as a precise standard for computer implementations of *Felt* so that different compilers will have similar behaviors. It also suggests an implementation strategy that can be adopted by compilers. Our *Felt* compiler adopts this implementation strategy while incorporating optimizations where beneficial. This approach simplified our task of implementing the compiler. Finally, the semantics aids in language documentation and analysis by assigning a precise meaning to every *Felt* specification. We believe this will encourage *Felt* users to develop correct specifications.

As mentioned above, *Felt*'s semantics serves as a formal specification for computer implementations of *Felt*. A *Felt* compiler can be verified to be correct with respect to this specification. In particular, *Felt*'s denotational semantics provides meanings to *Felt* specifications. Similarly, the PreScheme programming language's denotational semantics [8] provides meanings to PreScheme programs. We specify our *Felt* compiler to be correct if it maps *Felt* specifications to PreScheme programs such that the PreScheme programs have the same meanings as the *Felt* specifications.

We have chosen not to perform a detailed verification of the above statement (namely that the compiler preserves meanings) since the implementation strategy of the compiler is based on the denotational semantics. Rather, we have focused on the nontrivial algorithms of the compiler, namely the implementation of the constraints functions `is-a-word-in-list?` and `free-of-words-in-list?`. These functions are implemented as language recognizers (see Sec-

tion 4). We have formally verified the algorithms used to implement these functions; the rigorous mathematical proofs of correctness are presented in [10].

Our correctness proofs provide high assurance that the compiler does not alter the meaning (behavior) of the specified security filters. That is, the executable security filter programs generated by the compiler have the same behavior as that specified by the user. The formal semantics of *Felt* encourages users to develop correct specifications and provides some assurance that the specified behavior is indeed the desired behavior. Additional controls may be desirable to provide higher assurance of this fact. For example, modification (customization) of specifications can be restricted to certain individuals, good user interfaces can be developed, etc. Further work is needed to investigate these issues.

6 Conclusions and Future Work

We have developed a generic architecture for security filters. We have also developed a methodology for constructing security filters by specifying their behavior using a special-purpose specification language and then automatically compiling the specifications to executable code. A formal semantics for the specification language and verification proofs of non-trivial algorithms of the compiler provide for high assurance. We hope that this work will prove useful as a framework for developing security filters in the future.

Much work remains to be done. We have largely ignored the issue of user interfaces for different users of *Felt* (e.g., security filter implementors, security administrators, etc.). However, good user interfaces are critical to the usability of *Felt* as well as the assurance of the generated filters and they warrant considerable attention. Also, we have placed primary emphasis on developing the constraint sub-language of *Felt* at the expense of transformations and actions; these aspects need to be investigated further. Our claim of the broad applicability of *Felt* needs to be validated by specifying the behavior of several existing and planned security filters within *Felt*. Additional work is needed to integrate the generated filters into security guards. Finally, extensive empirical studies are needed to compare the performance and lifecycle cost of *Felt*-generated filters with custom-developed filters.

Acknowledgements

I thank Joshua Guttman for several suggestions, including the notion of a "security filter generator".

References

- [1] Bartlett, J. F., January 1989, *Scheme→C: A Portable Scheme-to-C Compiler*, WRL, 89/1, Digital Equipment Corporation Western Research Laboratory.
- [2] Fiorino, T., May 1991, *WWMCCS to CAT Guard, Functional Specification Baseline*, Project High Gear.
- [3] Gagnon, L., October 1990, *An Overview of the USAFE Guard*, In *13th National Computer Security Conference*, Baltimore, MD.
- [4] Guttman, J. D., V. Swarup, and J. Ramsdell, 1994, *The VLISP Verified Scheme System*, To appear in *Lisp and Symbolic Computation*.
- [5] Hopcroft, J. E. and J. D. Ullman, 1979, *Introduction to Automata Theory, Languages, and Computation*, Reading, MA: Addison-Wesley.
- [6] IEEE Std 1178-1990, 1991, *IEEE Standard for the Scheme Programming Language*, New York, NY: Institute of Electrical and Electronic Engineers, Inc.
- [7] Kelsey, R. A., 1992, *PreScheme: A Scheme Dialect for Systems Programming*, Submitted for publication.
- [8] Oliva, D. P., J. D. Ramsdell, and M. Wand, 1994, *The VLISP Verified PreScheme Compiler*, To appear in *Lisp and Symbolic Computation*.
- [9] Schmidt, D. A., 1986, *Denotational Semantics: A methodology for language development*, Newton, MA: Allyn and Bacon, Inc.
- [10] Swarup, V., November 1993, *Automatic Generation of High Assurance Security Guard Filters*, MTR 93B179, The MITRE Corporation.

BELIEF IN CORRECTNESS

Marshall D. Abrams, The MITRE Corporation, 7525 Colshire Drive, McLean, VA 22102,
abrams@mitre.org

Marvin V. Zelkowitz, Institute for Advanced Computer Studies and Department of Computer Science,
University of Maryland, College Park, MD 20742, mvz@cs.umd.edu

ABSTRACT

In developing information technology, you want assurance that systems are secure and reliable, but you cannot have assurance or security without correctness. We discuss methods used to achieve correctness, focusing on weaknesses and approaches that management might take to increase belief in correctness. Formal methods, simulation, testing, and process modeling are addressed in detail. Structured programming, life-cycle modeling like the spiral model, use of CASE tools, use of formal methods applied informally, object-oriented design, reuse of existing code, and process maturity improvement are also mentioned. Reliance on these methods involves some element of belief since no validated metrics exist. Suggestions for using these methods as the basis for managerial decisions conclude the paper.

1. INTRODUCTION

“Engineers today, like Galileo three and a half centuries ago, are not superhuman. They make mistakes in their assumptions, in their calculations, in their conclusions. That they make mistakes is forgivable; that they catch them is imperative. Thus it is the essence of modern engineering not only to be able to check one’s own work, but also to have one’s work checked and to be able to check the work of others.” [23]

Assurance is defined¹ as “the confidence that may be held in the security provided by a Target of Evaluation.” Informally, assurance is a “warm fuzzy feeling” that the system can be

relied upon to reduce residual risk to the predetermined level. Without delving into psychology, we observe that effectiveness and correctness both contribute to assurance. *Effectiveness* is determined by analysis of the specifications of the functional requirements; the environment in which the system will be used, the risks, threats, and vulnerabilities; and all the countermeasures, including physical, administrative, procedural, personnel, and technical. The system is considered effective if the result of this analysis is an acceptable residual risk. *Correctness* is determined by comparing the implementation of the countermeasures with their specification. The system is considered correct if the implementation is sufficiently close to the specification. Note that this definition of correctness is compatible with the concept of risk management and is closer to the concept of *trustworthy* than to *error-free*.

This paper shows how correctness can be established. All known methods contributing to correctness have shortcomings that make it impossible to establish correctness beyond reasonable doubt. That is, establishing correctness is a matter of belief, not proof. Under conditions of belief, we caution fiscal prudence in resources invested in assuring correctness. The major methods addressed in this paper are mathematical models, simulation, testing, process models and procedures. Minor methods, called silver bullets, include structured programming, the spiral model, Computer Aided Software Engineering (CASE) tools, formal methods applied informally, object-oriented (OO) programming, reusing existing code, and process maturity. Cost benefit is offered as a measure for selecting which belief system to embrace. We recommend hedging one’s investments by using more than one method.

¹ Definitions of *assurance*, *correctness*, and *effectiveness* are taken from the Information Technology Security Evaluation Criteria (ITSEC) [7]. Better definitions may be available by the time this paper is published.

Security-critical information technology (IT) systems² are extremely dependent on correctness. In systems involving human life and safety, correctness is paramount. A security-critical IT system must do exactly what is identified in its specification and not do anything that is not so specified. Correctness of software always has to be *with respect to a specification*.

Various methods may be used to demonstrate correctness, but all are less than perfect and involve some element of belief in relying on the results of using that method. That is, it cannot be proven that a method is "good" or "better." The methods are complementary in contributing to correctness itself as well as in contributing to belief in correctness. There is a growing consensus that, to say the least, no one technique can provide adequate assurance (see, for example, [5]. David Parnas [22], among others, has suggested that an "assurance tripod" is required: the combination of rigorous testing, evaluation of the process and personnel used to develop the system, and a thorough review and analysis of various products produced during development. In the pragmatic end, managerial judgment determines resource allocation to correctness and assurance. In this paper, we focus on practical product correctness and the various problems one has in achieving this correctness.

We should learn from branches of natural science and engineering that have been trying to understand complex systems far longer than computers have existed. One important objective is to recognize when simplifying assumptions are valid and when they are dangerous. One of the authors learned as a sophomore that "the essence of engineering is to make enough assumptions so that you can solve the problem, without assuming the problem away."

Let us consider whether formal theories of programming are good approximations of real programs executing on real computers. Although the theories are relatively simple, applying them

to realistic programs vastly complicates the model. You cannot even assume simple axioms like "For all integers i , $i+1>i$ " on fixed wordsize computers since integer i may "overflow" and have an unspecified, negative, zero, or the same value, depending upon the particular hardware executing the program. Mathematical models of computer programs generally do not accurately represent the subtlety of programs in an environment (i.e., execution on real hardware). The mathematics of computer modeling belongs in the realm of applied rather than pure mathematics.

When we use Ohm's law, Kirchoff's rules, etc., to design an electronic circuit or use Newton's laws to predict the orbit of a satellite, no one is saying that they have "proved" that the circuit works or that the satellite will be exactly where they said it would be. By the same token, when we model a computer program using some method such as Hoare's [13] we then have some confidence (maybe little) that the program when executed will behave much as we predict (but perhaps not exactly like we predict—e.g., integer overflow). This requires that even simple programs have complex proofs in order to show that the mathematical properties of the program behave as desired. Simple formalisms for programs are too complex to accurately represent most programs in execution on physical machines.

This insight shows that formalisms in programming are very different from formalisms in the natural sciences. In the natural science, you have a theory (e.g., Laws of Motion) that is a good approximation to the physical interactions among objects. In physics, a sufficiently accurate approximation gives useful results. In contrast, for programming, you must approximate the program and the hardware (e.g., assume integers are infinite) in order to have any relationship to the formal model. A key difference is *lack of continuity*. In programming, disastrous examples of integer overflow and other discontinuities show that the supposed approximations are not necessarily close. Use of discrete logic to model these leads to expressions of enormous complexity [21]. Alternatively, models could incorporate known characteristics and limitations of the computer to increase their veracity. We do not wish to compare good models of physics with bad models of computers. Newton's laws do

² The term *IT system* includes all sizes of computer systems, from super mainframes to desktop units to embedded components and controllers, as well as networks and distributed systems.

not work well for objects at near the speed of light or for objects that are not in inertial frames of reference. Likewise, a Hoare model of computer system behavior is a poor representation if the integer values are at or near the overflow. One would need to modify the model to accommodate the overflow behavior. Once having done so, the model would be better.

Several methods have been developed and been accepted over time to demonstrate the correctness of computer programs. None of these heuristics are true in the sense that they portray absolute infallibility of the method. Each has proponents and detractors. In the next section, we describe these methods, explore ways in which each accomplishes its task, and draw some conclusions from this analysis.

2. CORRECTNESS METHODS

Several techniques are regularly employed to show that a computer program does exactly what it is supposed to do and nothing else. The first two described below, formal methods and simulation, analyze the program and derive properties about it. The third, testing, experiments with program behavior, perhaps using some information derived by application of the first two techniques. The fourth technique, process models and procedures, looks at the development process itself under the assumption that good development practices result in good software.

Each method is described briefly, emphasizing its advantages, disadvantages, and contributions to our belief system. A common distraction with all methods is the complexity of execution. The steps, processes, or manipulations that constitute the practice of the method can be so overwhelming that perspective is lost. We agree with Hamming [12] that "the purpose of computing is insight" and that it is difficult to retain perspective and insight in the face of complexity. It is very easy to get caught up with all the mechanics of employing a method so that in practice the mechanics get emphasized at the expense of understanding.

"When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfac-

tory kind" [14]. Metrics of correctness need to be developed and applied to individual methods and combinations of methods. We need to replace belief with analysis if at all possible. While early work on the Capability Maturity Model [19] and the Experience Factory [1] show that we may develop such metrics, more needs to be done.

2.1 Formal Methods

The use of formalisms stems from two related observations: natural language tends to be imprecise, and in achieving precision, there is the potential for automation. Mathematical notation has the advantage of precision and is associated with rigorous, logical thinking that assists in reducing ambiguity. In principle, formal models of IT systems can support all phases of the system development process: articulation of policy for use, high-level architecture, design, and implementation. Today, formal models of security policy help perfect understanding and development, especially of new policies. While formal specifications are used in Europe, they have not made much of an impact in the United States. No language is likely to be a cure-all in achieving higher levels of abstraction, and more natural models of problem spaces, for all problem spaces.

In discussing formal methods, we have to be sure to differentiate them from formalized methods, such as Computer Assisted Software Engineering (CASE) tools, structured analysis, and other mechanized methods for developing source programs [25]. In using formal methods, one traditionally begins with a formal description of the specification of a software system according to some underlying mathematical model and the realization of that specification as a concrete design or source code implementation. (Other possibilities are to start with a description of how the system is to be used, or to let an automated deduction system participate directly in the construction of later design and implementation stages.) Using mathematical principles, one shows that the program agrees with the model. For example, axiomatic verification, perhaps the oldest of the formal techniques, assumes we have a program S , a precondition (specification) P that is true before the execution of S , and a postcondition (output specification) Q . We need a proof that demonstrates: (1) the relationship among S , P , and Q that determines the effect

program S has on P to assure that Q will be true after execution terminates, and (2) program S does indeed terminate if P is true initially [13]. If we derive a set of axioms for each statement type in our language (e.g., rules for describing the behavior of the if statement, the while statement, the assignment statement), then we have tied program correctness to the problems of generating correct mathematical proofs. But we still have not proved that the program when executed on a specific computer is correct because of the very problems raised earlier. At best we have shown that the formal description of the program satisfies its specification (i.e., produces the given post condition when the precondition is true) [10]. Our confidence in the correctness of the program is dependent on our confidence that our formal model is an accurate representation of the target computer.

As described previously, when we use formal models we need to suppress details to make the models tractable. Unfortunately, many of the details suppressed in the formal models are implementation dependent and security relevant. Formal models are losing ground to the complexity of networked and distributed systems. It is difficult to scale up the traditional use of formal methods to large complex systems. While they may appear to work satisfactorily on small "toy" problems, there has been little evidence that they scale up very well [21].

"Larger examples are necessary to demonstrate how these concepts scale up" [30]. Formal models are often applied to complex systems combined with other belief systems. For example, variants of the Bell-LaPadula security policy model [2] are often cited as the basis of operating system security, but the actual implementations also include security-relevant processes, called *trusted* or *privileged*, that are not formally modeled. Belief that security is preserved after introduction of these processes is often established by non-formal means. Practitioners of formal modeling sometimes appear to forget about the assumptions and simplifications that were made to make their models tractable and fail to caveat the applicability of their results to the real world. This is an error on the part of the practitioners. A great deal of the simplifying assumptions are made because the modelers simply do not know how to model some of these features (although

many are certainly susceptible to being modeled), or the resources available do not permit modeling the necessary details. Perhaps the practitioners are not experienced enough in this kind of mathematics.

Within the limits imposed by the simplifications and assumptions made for the sake of tractability, formalism can be used both to determine correctness of the implementation and adherence of the system to certain properties. We can prove that a given procedure must return a certain value and also show that certain policies are never violated. Many observers *believe* that formal policy models have their maximum benefit in removing inconsistencies, ambiguities, and contradictions in the natural language policy statement. The *process* of formalizing the policy aids in clarifying the policy. This process then has the secondary benefit of making a clearer statement of policy to the implementors.

Although formal methods are based on mathematical proofs, we must realize that even mathematical proofs may have flaws. "Outsiders see mathematics as a cold, formal, logical, mechanical, monolithic process of sheer intellection... [but] Stanislaw Ulam estimates that mathematicians publish 200,000 theorems every year. A number of these are subsequently contradicted or otherwise disallowed, others are thrown into doubt, and most are ignored. Only a tiny fraction come to be understood and believed by any sizable group of mathematicians" [9]. Although mathematicians do not like to admit it, correctness can be likened to a social process—it is only the test of time where no flaw has been discovered that builds our confidence in the ultimate truth of a theorem. All scientific processes have flaws. Petroski [23] argues that failure is an important part of engineering design. It is only when things fail that we understand how to make them better. How well would we be designing bridges if none ever collapsed? Either we have overbuilt them to a point of economic stupidity, or we have never stressed them sufficiently. We hope that by our continuing (unsuccessful) attempts to model computers, we are learning something.

2.2 Simulation

Simulation is the development of a simplified version of a system's specification by eliminating non-critical attributes to develop a system

that exhibits relevant properties. By ignoring certain properties, it is often possible to quickly and inexpensively build simpler versions of a system. Using this simulation, security-related principles can be more readily developed and examined. This increases our belief in the ultimate specification since we have demonstrated the existence of an implementation that already has the desired properties.

While we can simulate a system to test the security policies, the interaction of these policies with the assumed-away specifications of the complete system severely lowers our belief in the correctness of the overall system with respect to security. By definition, one is “abstracting away” non-essential aspects of the system when doing simulation and modeling—yet it is very hard to develop “non-interference proofs” for those missing aspects, so that you have confidence that they really won’t change the behavior of interest in the “real” system. It is only by testing (and/or formalism) applied to the complete system that adds to our belief in this product—although the existence of a simulation that implements our security policy does provide a sort of existence proof on policy and increases our confidence (i.e., belief) in a complete implementation. (See spiral model discussion, below.)

2.3 Testing

Testing demonstrates behavior by executing a system using a selected set of data points to show that the system executes correctly on those points. The assumption is made that if the set of data points is chosen appropriately, then the behavior of the system for most data points will be analogous to the selected data points. If we believe that the selected data points are representative of the domain of data in which we are interested, we have confidence in the correctness of our implementation. Choosing the selected data points and the best method of testing our program are our major decision steps toward determining our belief in the correctness of this system. Knowledge gained from formal methods, code analysis, and simulation can help focus the selection. As pointed out by Leveson [16], “testing researchers have defined theoretical ways of comparing testing strategies both in terms of cost and effectiveness (for example, [29]), formal criteria for evaluating testing strategies (for example, [11]), and axioms or

properties that any adequacy criterion (rule to determine when testing can stop) should satisfy (for example, [28]).” Analytic results can also indicate when statistically significant measurement results have been obtained [17].

Testing methods can be divided into functional, performance, failure-mode, and, for security, penetration. Functional testing includes testing against a catalog of flaws previously discovered in this or other systems. The major thrust of security testing is in penetrating (i.e., violating the security policy), thereby measuring the resistance to anticipated threats. The presence of anticipated threat actions, possibly by a malicious adversary, distinguishes the security concerns in a system.

Testing functional specifications is usually achieved by black-box testing, in which the tester only has access to the specifications of the program, while testing specific program behavior by understanding the design is achieved by glass-box (a.k.a. white-box) testing, in which the tester has access to the internal source code of the program. Security testing of high-assurance systems proceeds with extensive documentation of design and implementation. Varying degrees of assurance are obtained according to the information available to the testers, including security kernel code, design documentation, and formal models. The value of penetration testing depends on the experience of the testers and the methodology employed. IV&V (Independent verification and validation), where a group independent from the developers is charged with testing a system, is sometimes effective in finding errors that developers who are too familiar with the source program may overlook. As with many of the methods addressed in this paper, the cost-benefit of this added level of assurance must be analyzed [20].

The classical example by Dijkstra shows that exhaustive testing cannot prove correctness of any implementation. To prove the correctness of “ $a+b=c$ ” on 32-bit computers would require $2^{32} \times 2^{32} = 2^{64}$ or over 10^{19} tests. At a rate of even 10^8 tests per second, that would require 10^{11} seconds or over 3,000 years. Perhaps we should ask ourselves whether we really have so little understanding of the operation of a computer that we have to test addition, for example, for all possible addends to be convinced that the

addition function is working correctly? Under what conditions can we state a general argument that works in the face of overflow? Although it is recognized that testing cannot be exhaustive, testing has a very strong intuitive appeal and constitutes a very strong basis for belief in correctness.

Testing always involves comparing the actual results of execution with anticipated results. One way to capture anticipated results is to test an executable specification of a prototype. Once this is done, it is possible to automatically execute the system being tested and its specification in parallel, and to automatically compare the results, thereby greatly increasing the number of feasible test cases [27].

2.4 Process Models and Procedures

All of the previous techniques depend upon subjecting a program to one of the discussed methods to increase confidence that the program exhibits correct behavior. However, as we have often stated, this is extremely difficult to do. As an alternative, perhaps it is easier to understand the mechanisms used in developing the program under the belief that correct methods yield correct programs. The idea underlying process models is that understanding what you are doing is a necessary step to improvement. By using a simple, well understood process to develop software, we have belief that the ultimate product best meets our needs. Two process models currently enjoy favor: waterfall and spiral. The United States Department of Defense (DOD) standards imply (but do not require) use of the former in management of software development.

The *waterfall model* [24] conceives of software development as a linear process based upon a set of deliverable artifacts. There are easily recognized milestones between steps in the process. Although the mechanisms of the process are generally obscure—only the results of the process are visible. Therefore, the waterfall model uses these products—a specifications document, a design document, a source file, and the results of testing, for example. These milestones can support a management strategy of schedules and reviews. Recognition that the process is not perfect led to the introduction of feedback paths in the model. If drawn as a waterfall of steps, the feedback paths suggest salmon swimming upstream. The feedback paths

represent knowledge gained in latter steps that affect activities and decisions made earlier. It may be necessary to adjust, or even abandon, earlier work as a consequence of feedback. In practice, schedules tend to not allow for such corrective action. Non-technical project managers are often determined to meet their schedules, no matter what the consequences [26].

Because of all of these deficiencies, belief in the waterfall model as a useful methodology for developing software that satisfies its specification has been slowly decreasing, and an alternative *spiral model* has been gaining favor [3]. The spiral model emphasizes the process of developing software rather than the resulting products. It is also called a *risk-reducing model*, since the basic premise is to develop and prototype a solution, evaluate the risks of adding specifications, and repeat the process. Each cycle of the model creates a more complex version of the system, with the ultimate prototype being the final system itself. At each stage, we use Occam's razor to simplify our solution, we make the process of development as visible as possible, and we try to quantify the risks involved in continuing development. Thus, our belief in the solution should be higher than with the hidden processes inherent in the waterfall model. The spiral model emphasizes the repetition of basic activities at progressive stages of a project. The exact activities change as the project matures, but such activities as design, implementation, testing, evaluation, and planning are related. Changing requirements are more easily accommodated. The cost is represented by the radial distance in a polar coordinate system and the activities occur at a specified polar angle. Progress is assumed proportional, or at least related to, cost. While the theory of the spiral model accommodates redesign and backtracking, the imposition of schedules can have exactly the same effect as on the waterfall model.

3. CHOOSING AMONG ALTERNATIVE BELIEFS

Software engineers promote one technique after another as the "silver bullet" [4] solution to all our problems. This section examines the most popular silver bullets.

3.1 Tarnished Silver Bullets

To address correctness in system development, many techniques have been proposed as potential solutions (e.g., see [6, 8]). All techniques involved a measure of belief as groups of professionals argued among themselves regarding the appropriateness of their favorite method. None has completely provided the warm fuzzy feelings we want :

- a. Structured programming (e.g., “goto-less programming” of the 1970s) makes programming easy and correct. Twenty years of experience have shown that quality has improved, but not to the level initially proposed. There *is* a relationship between the restrictions imposed by using only the appropriate control structures and formal verification of the source code produced; however, errors still occur in such programs [31].
- b. The spiral model is superior to the waterfall model. The spiral model was an improvement in that it emphasized the process of software development with attendant interest in the management, risk evaluation and reduction, and prototyping aspects of the process. Note that this is an example of Petroski’s theses. Because the waterfall methodology has proved inadequate to produce good software, a new methodology (spiral) has been introduced. When it is determined that the spiral also is inadequate, creative people will develop a new system. Since we do not have good measures of correctness, it is difficult to know how to make the process better. Note also that the spiral model and the waterfall model that it replaced both represent a similar set of practices as actually implemented by many organizations.
- c. CASE tools will supplement the intelligence lacking in today’s programmers. Unfortunately, the tools have not added much intelligence and today’s programmers could still use additional help. Case tools suffer from the same problem as the other software we are discussing: they have errors (all software has errors), and they are only as smart as their developers.
- d. Formal methods applied informally (e.g., languages like VDM and Z) can improve the process. While this seems to be true, it has yet

to be demonstrated that this approach results in the correctness that we need for security-related systems. It is not clear that our belief in these specification techniques will be high enough to eliminate the need for alternate mechanisms. Nor is it clear that our beliefs are the only ones that count. See [18] for a discussion of mathematical arguments that qualify as proof in a court of law.

- e. Object-oriented (OO) programming and design will replace conventional design techniques, and languages that implement such processes (e.g., C++) will replace other languages (e.g., Pascal, Ada, FORTRAN). This concept represents one of the newer trends in program design. We do not have enough evidence to judge the effects of OO design on security. This technique does encapsulate some of the formal data-structuring mechanisms into the programming language; however, it must still be observed what effects it will have on overall system correctness. (Note that this is just the current version of the traditional silver bullet, “Language X will make programming easier.” In the 1960s, we had COBOL and then PL/I, in the 1970s we had Pascal, in the 1980s we had Ada, and now we have C++.). Each language is perceived to have failed in achieving some objective. Hence, someone develops a new language to correct the flaws. This cycle will probably never end, as it is not likely that any one language will be perfect for all applications.
- f. Reusing existing code is the solution. Since code proven correct once need not be so proven again, one only needs to create a library of reusable components. While reusing existing code is an admirable goal, we still do not have the technology to implement this process. While we can create write-only libraries of reusable components, we have no process available that enables us to determine the specifications of an existing library component and whether it fulfills the specifications for another application. Current interests in domain-specific architectures and faceted classification schemes are both attempts at understanding the functionality of reusable components. We reuse hardware components all the time, in the sense that we manufacture identical copies of circuit packages and other components. Each component conforms to

some specification of performance and behavior that is described in components manuals. Why can't we do something similar with software?

- g. Process maturity improvement is today's salvation [19]. Current thinking is that improving only the process without looking at the ultimate product being produced is all that is necessary to produce quality software. While it should greatly improve the production of software from many organizations that currently have *no* such process, as shown often in the past, this is a naive approach to producing correct software.

We do not mean to say that the above techniques are failures. All, to some extent, improve upon the quality and correctness of the resulting program that is produced. Programming as taught in the universities and practiced in industry today is radically different from that of the 1960s. However, the important point is that none of them achieves the level of correctness that would support our belief in that technique over all others.

3.2 Which Belief System to Embrace

Resources must be allocated among the correctness methodologies. While management has been described as the art of making decisions based on inadequate information, the quality of decisions is often improved by providing more information. Installation and use of security-critical IT systems cannot wait for proofs of efficacy or development of metrics for determining cost-benefit. Managers will need to continue to make decisions whether or not to employ IT. The managerial authorization and approval granted to an IT system to process sensitive data in an operational environment is, in theory, made on the basis of analysis and certification of the extent to which design and implementation of the system meet pre-specified requirements for achieving adequate security. Security objectives can be met by a combination of technical means within the system and physical and procedural means outside the system. In this theory, when management accredits the system, management is accepting the residual risk.

How can we address this residual risk? While we have no clearly defined metric for this, we do have examples of systems that seem to ade-

quately address our security concerns. One avenue of research is increased study of these "artifacts"—the systems, designs, and specifications that have helped produce acceptable solutions. This knowledge should enable us to produce better models in the future. However, today there is no way to measure the residual risk, nor is there a metric for cost-benefit. So, how is a decision made? Since computer and management science cannot help verify a decision, the experienced manager's intuition cannot be dismissed. Experience probably includes comparison with previous efforts, the correctness of which has become better known over time. One must be careful to distinguish between management saying "I did this before and it worked" versus "I feel safe using this since I used it before, while this new technique is unknown to me." The first statement encapsulates the experiences of good management, while the second statement reinforces unscientific prejudices. The real problem is how to differentiate among good science, common sense, and stubborn stupidity.

In deciding which belief system to embrace, the prudent manager probably hedges by using more than one system. Various combinations of formalism, testing, simulation, and process may be employed. Since cost is one of the attributes we need to address in evaluating the overall quality of the product, it is prudent that management should adequately choose from among the techniques those that meet required cost constraints yet still meet functional requirements for the product.

4. RECOMMENDATIONS

Given the absence of metrics for any of the belief systems, the inherent difficulty in using any of them, and the lack of a repository of correctness artifacts to study and evaluate, the authors do not propose to solve this problem with a pronouncement of correct technique. Our focus is to increase the awareness of the technical and managerial segments of the IT security community to the limitations of each of these techniques. We attempt to increase understanding of the need to address more than one solution to the multifaceted correctness problem.

We view the glass as being half full. We do not advocate that anyone abjure his belief(s) in correctness. Rather, we suggest that attempts to

prove beliefs are bottomless pits. Unless some breakthrough occurs, we advocate treating this aspect of software engineering pragmatically. Just as engineers built steam engines (see [16] for further analogy) before the science of thermodynamics was developed, the software engineering community can build software systems based on intuitive and pragmatic notions of how to attain correctness and other aspects of quality. At least now, we should acknowledge practicing an empirical discipline.

At the risk of appearing cautiously optimistic, we hesitantly endorse four interrelated strategies. The exact allocation of resources among the strategies remains a technical management decision. Looking at the mature methods available today, we tend to agree with the perceived consensus that a combination of the following should be employed:

- Evaluation of process, personnel, and abilities to identify and reinforce positive attributes
- Thorough review and analysis of intermediate products during development with sufficient time and resources allocated to correct deficiencies
- Rigorous testing based on the preceding analysis
- Recognition of critical points in system development
 - Point of diminishing return for application of any method
 - When a development should be terminated for cause or to stop hemorrhaging

Looking forward, we see promise in combining aspects of program reuse and object orientation. The possibility of employing object self-protection in security architecture should be considered.

Each of the techniques described in this paper has an aspect that help increase our belief in the correctness of an implementation, yet each is fraught with some dangers. Each technique comes with some, generally high, cost for its use. It is imperative that management addresses each as aids in developing security-critical IT systems and not arbitrarily dismiss any of them. We should:

- Be cognizant of the limitations of each

- Belief in correctness should be relative
- Be prudent in establishing realistic assurance requirements for a given system that are measurable, achievable, and cost-effective
- Resist the temptation of unachievable elegance and perfection
- Differentiate between research and operations
 - Define achievable specifications
 - Accept residual risk

5. ACKNOWLEDGMENTS

We appreciate the contributions from the following individuals on previous drafts of this paper: Rochelle Abrams, Sharon Fletcher, Lester Fraim, John Gannon, David Gomberg, Ronald Gove, Chuck Howell, Jay Kahn, Carl Landwehr, John McLean, Jonathan Millen, Jonathan Moffett, Jim Purtilo, Jim Williams, John P. L. Woodward, and the anonymous reviewers. Research support on this activity for Marshall Abrams was provided by the National Security Agency under contract DAAB07-94-C-H601, and for Marvin Zelkowitz was partially provided by NASA grant NSG-5123 from NASA/Goddard Space Flight Center to the University of Maryland.

6. REFERENCES

1. Basili, V. R., G. Caldiera, and G. Cantone, 1992 "A Reference Architecture for the Component Factory," *ACM Transactions on Software Engineering and Methodology*, Vol. 1, No. 1, pp. 53-80.
2. Bell, D. Elliott, and Leonard J. LaPadula, April 1974, *Secure Computer Systems: Unified Exposition and MULTICS Interpretation*, MTR 2997, The MITRE Corporation, Bedford, MA. Available from National Technical Information Service, AD/A 020 445.
3. Boehm, B., May 1988, "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, Vol. 21, No. 5, pp. 61-72.
4. Brooks, F., 1987, "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, Vol. 20, No. 4, pp. 10-19.
5. Butler, R. W., and G. B. Finelli, 12 January 1993, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Transactions on Software Engineering*, Vol. 19, No. 1, pp 3-12.
6. Chang, C., 5 September 1993, "Is Existing Software Engineering Obsolete?," *IEEE Software*, Vol. 10, No. 5, pp. 4-5.

7. Commission of the European Communities, 28 June 1991, *Information Technology Security Evaluation Criteria (ITSEC): Provisional Harmonized Criteria*, Luxembourg: Office for Official Publications of the European Communities, Version 1.2.
8. Davis, A., 5 September, 1993, "Software Lemmingengineering," *IEEE Software*, Vol. 10, pp. 79-84.
9. DeMillo, R., R. Lipton and A. Perlis, May, 1979, "Social Processes and Proofs of Theorems and Programs," *Communications of the ACM*, Vol. 22, No. 5, pp. 271-280.
10. Fetzer, J. H., September 1988, "Program Verification: The Very Idea," *Communications of the ACM*, Vol. 31, No. 9, pp. 1048-1063
11. Goodenough, J. B., and S. Gerhart, June 1975, "Toward a Theory of Test Data Selection," *IEEE Transactions on Software Engineering* Vol. SE-1, No. 2.
12. Hamming, R., 1962, *Numerical Methods for Scientists and Engineers*, McGraw Hill.
13. Hoare, C. A. R., October, 1969, "An Axiomatic Basis for Computer Programming," *Communications of the ACM*, Vol. 12, No. 10, pp 576-583.
- _____ August 1986, "Mathematics of Programming," *Byte*, pp 115-121.
14. Kelvin W. T., 1881-1884, *Popular Lectures and Addresses*.
15. Knight, J. C. and D. M. Kienzle, 1992, "Preliminary Experience Using Z to Specify a Safety-Critical System," *Proceedings of 1992 Z Users Workshop*, Springer-Verlag.
16. Leveson, N. G., May 1992, "High-Pressure Steam Engines and Computer Software," *Proceedings International Conference on Software Engineering*, Melbourne, Australia.
17. Mamrak, S. A. and M. D. Abrams, December 1979, "A Taxonomy for Valid Test Workload Generation," *Computer*, pp. 60-65.
18. MacKenzie, November 1992, "Computers, Formal Proofs, and the Law Courts," *Notices of the American Mathematical Society*, Vol. 39, p. 9.
19. Paulk, M. C., B. Curtis, M. B. Chrissis and C. V. Weber, July 1993 "Capability Maturity Model for Software, Version 1.1," *IEEE Software*, Vol. 10, No.4, pp. 18-27.
20. Page G., F. E. McGarry and D. N. Card, June, 1985, Evaluation of an independent verification and validation methodology for flight dynamics, NASA/GSFC Technical Report SEL 81-110.
21. Parnas, D. L., December 1985, "Software Aspects of Strategic Defense Systems," *Communications of the ACM*, Vol. 28, No. 12, December 1985, pp. 1326-1335.
22. Parnas, D. L., A. John van Schouwen, and Shu Po Kwan, June 1990, "Evaluation of Safety-Critical Software," *Communications of the ACM*.
23. Petroski, H., 1985, *To Engineer is Human: The Role of Failure in Successful Design*, St. Martin's Press.
24. W. W. Royce, 1970, "Managing the Development of Large Software Systems: Concepts and Techniques," *Proceedings IEEE Wescon..*
25. Rushby, J., December 1993, *Formal Methods and the Certification of Critical Systems*, Technical Report C3L-93-7, Stanford Research Institute.
26. Stillman, R., March 22, 1993, "Software Development: Neither Economics nor Engineering," keynote address, Third Annual Software Engineering Economics Conference, The MITRE Corporation.
27. Taylor, T., October 1989, "FTLS-Based Security Testing for LOCK," *Proceedings of the 12th National Computer Security Conference*, pp 136-145.
28. Weyuker, E. J., December 1986, "Axiomatizing Software Test Data Adequacy," *IEEE Transactions on Software Engineering* Vol. SE-12, No. 12, pp. 1128-1138.
29. Weyuker, E. J., S. Weiss, and D. Hamlet, October 1991, "Comparison of Program Testing Strategies," *Proceedings of the Fourth Symposium on Software Testing, Analysis and Verification (TAV4)*, Victoria, B.C., Canada, pp 1-10.
30. Youngblut C., et al, February 1989, *SDS Software Testing and Evaluation: Are View of the State-of-the-Art in Software Testing and Evaluation with Recommended R&D Tasks*, Institute for Defense Analysis Report p. 2132.
31. Zelkowitz, M. V., November 1990, "A Functional Model of Program Verification," *IEEE Computer*, Vol. 23, No. 11, pp. 30-39.

TOWARDS A PRIVACY-FRIENDLY DESIGN AND USE OF IT-SECURITY MECHANISMS

Simone Fischer-Hübner
University of Hamburg
Faculty for Informatics
Vogt-Kölln-Str.30
D-22527 Hamburg, Germany
e-mail: fischer@rz.informatik.uni-hamburg.d400.de

Abstract:

Security mechanisms can be used to protect personal data from misuse. However, the additional level of control may also endanger privacy. This is the conflict between security and privacy. Besides, there is the problem that today's security mechanisms are mostly not appropriate to enforce basic privacy requirements. This paper presents a formal task-based privacy model that can be used to technically enforce legal privacy requirements in an operating system and a holistic approach towards a privacy-friendly use of security mechanisms to cope with the conflict between security and privacy.

1. Introduction

IT-Security Mechanisms are often seen as technical data protection measures, i.e. technical measures to protect privacy. Unfortunately, often it is not considered that there is a conflict between privacy and security, as security mechanisms on the other hand are control mechanisms that often have to collect and process personal control data about the users that can be misused e.g. for performance monitoring. Furthermore, today's security mechanisms mainly address the confidentiality of (classified) information, but not basic privacy principles. The wellknown Bell LaPaduala model [Bell LaPadula 76] for example, which is the basis for the Orange Book [TCSEC 85] and the example functionality classes of ITSEC [ITSEC 91], is not appropriate for enforcing privacy requirements, such as *purpose binding* or *necessity of data processing*.

In this article, these problems and misconceptions are discussed. Besides, it is shown how security mechanisms can directly address privacy requirements. For this purpose, a task-based formal privacy model is introduced that can help to technically enforce legal privacy principles in an operating system. Furthermore, to protect personal control data produced by security mechanisms and thus to bring security and privacy nearer, a holistic approach towards a privacy-friendly use of IT-Security mechanisms is presented.

2. Privacy Requirements

In the information society, privacy is accepted as an important personal right needing protection. It can be defined, as it has been done by the *German Constitutional Court* in its Census Decision of 1983, by the term *right of informational self-determination*, meaning the

right of an individual to determine about the disclosure and use of her/his personal data on principle at her/his discretion.

In order to protect this right, privacy laws of most western countries, e.g. the German Data Protection Act (Bundesdatenschutzgesetz, BDSG) or the U.S. Privacy Act, require basic principles to be guaranteed when personal data are collected or processed, such as:

- ***purpose binding*** (personal data obtained for one purpose should not be used for another purpose without informed consent, e.g. § 14 BDSG)
- ***necessity of data collection and processing*** (the collection and processing of personal data shall only be allowed, if it is necessary for the tasks falling within the responsibility of the data processing agency, e.g. §§ 13, 14 BDSG),
- ***requirement of adequate technical and organisational safeguards*** (e.g. § 9 BDSG) to guarantee the confidentiality, integrity and availability of the personal data.

3. Conflict Between Security and Privacy

Technical security mechanisms are necessary to protect personal data against accidental or unauthorized access, modification or other illegal processing. Security mechanisms are required by many privacy acts, because they are regarded as technical means to protect privacy. Unfortunately, as already mentioned, the contrary can also be the case:

Security mechanisms often require the collection and use of personal control data about users and usees (a usee is a person who is personally affected by the data collected and processed about her/him, but who has no control over this process). These personal control data can be misused for example for performance monitoring. This results in a conflict where security mechanisms can both help to protect the privacy of data subjects and at the same time can be used to invade the privacy of users and usees (see [Denning et al. 87], [Fischer-Hübner/Yngström/Holvast 92], [Ketelaar/Fischer-Hübner 93]).

The following examples demonstrate how security mechanisms endanger the privacy of the system's users, about whom personal data are collected and who are therefore at the same time usees, and of other usees, who are not actively involved [Fischer-Hübner/Yngström/Holvast 92]:

Authentication mechanisms, especially continuous authentication, such as keystroke dynamics, can produce information about the user's presence. Furthermore, if devices such as smartcards are used as employee authentication for access to certain security relevant areas, the employee's movements and her/his contacts with other employees can be monitored.

Discretionary access control mechanisms (DAC), such as access control lists, require the storage and usage of information about the user's access rights. **Mandatory secrecy and integrity access controls (MAC)** use security and integrity levels, that are attached to users and to objects. These access control data reveal personal information about the user's status. Furthermore, in order to attach a security or an integrity level to a user, her/his trustworthiness has to be checked e.g. by surveillance. That means an additional limitation to the user's privacy.

Auditing produces information about the activities and behavior of the users. If activities of users with respect to other people (uses) are audited, the audit trails may also contain critical information about these uses.

Monitoring of unusual and security relevant activities on a system through system status information also means a continuous control of a user's actions.

Intrusion Detection Expert Systems that are based on Denning's Intrusion Detection Model [Denning 86] produce and use statistical profiles that store statistics about the behavior of subjects (e.g. users) with respect to objects. These statistics can be directly misused for employee performance control [Denning et al. 87].

Besides these security mechanisms that control the actions of users **backup-mechanisms** can also endanger privacy rights. Backup files can contain personal data, that were stored on the system at backup-time, but have become outdated and incorrect. According to most privacy acts, individuals have the right to have incorrect personal data corrected and illegally stored data deleted. Normally such corrections are only done on system's on-line data and not on the backup-files. The consequence is that after the correction or deletion of personal data in the system, backups may still store the incorrect personal data and may therefore be in conflict with the data subject's privacy rights.

4. Technical Enforcement of Privacy Requirements

There is also another reason why today's security mechanisms are mostly not appropriate technical means to enforce privacy: They mainly enforce confidentiality, and partly integrity and availability policies and controls, but they do not directly address basic privacy requirements as listed above.

Having been funded by the U.S. Government, research and development of secure systems have been mainly concentrated on maintaining secrecy of classified information. Systems have been preferably developed to enforce Mandatory Access Control (MAC) and Discretionary Access Control (DAC) as required by the classes of the Orange Book [TCSEC 85] and by the example functionality classes of ITSEC [ITSEC 91].

MAC (as defined by the Orange Book) restricts the access to objects based on the sensitivity of the information contained in the objects and the clearance of the subjects. But the problem is that personal data cannot be classified accurately by its sensitivity *per se*, because the sensitivity of personal data is related to the purpose and context of its use. In its Census Decision the *German Constitutional Court* proclaimed that there are no non-sensitive personal data, as dependent on the purpose and context of use all kinds of personal data can become sensitive. There are personal data that *per se* already contain sensitive information (e.g. medical data), but dependent on the purpose and context of use, such sensitive data can become even more sensitive and data that seem to be non-sensitive (e.g. addresses) can become highly sensitive as well. In order to enforce privacy, it should be checked whether the purpose of the task, currently performed by the user who wants to access personal data, corresponds to the purpose for which that personal data were obtained (**requirement of purpose binding**).

DAC restricts access to objects based on the identity of subjects and/or subject groups. DAC permits the granting and revoking of access privileges to data to be left to the discretion of a user with a certain access permission that has control over the data. But under privacy aspects, personal data about a data subject should not be "owned" or "controlled" by another person. In order to protect privacy, an access control decision should not be determined by the user's identity, but by the task that the individual user is currently performing. Personal data should only be accessible to a user, if such access is necessary to perform his/her current task and if he/she is authorized to perform this task (**requirement of necessity of data processing**).

5. A Formal Privacy Model

In this section, the concept of a formal security model for operating systems that directly enforces basic privacy requirements, such as *purpose binding* or *necessity of data processing*, shall be introduced in more detail.

5.1 Formal Description

The **privacy policy** that the model shall enforce can be described informally as follows:

A user shall only have access to personal data, if this access is necessary to perform his/her current task and only, if the user is authorized to perform this task. Besides, the purpose of his/her current task must correspond to the purpose for which the object was obtained or there has to be consent by the data subject.

This **formal task-based privacy model** contains the following *state variables*, *invariants* (*privacy properties*) and *state transition functions*:

a.) State variables

First, the security-relevant (or better: privacy-relevant) state variables shall be defined that are needed to formally define the privacy policy and the system states.

Subjects S: Subjects are the active entities of the system.

S = set of current subjects = {S₁, S₂,...}

Objects O: Objects are passive entities containing personal data.

O = set of current objects containing personal data = {O₁, O₂,...}

Personal data are data about an identified or identifiable person. The question whether a person is identifiable also depends on the additional knowledge that a potential attacker has. As this additional knowledge cannot be known, all data should be considered as personal data, if the possibility of reidentification cannot be excluded in practice.

Tasks T: A subject shall be allowed to access an object only by performing a task. The tasks have to be defined for each application.

T = set of tasks = {T₁, T₂,...}

Current Tasks CT: The task that is currently performed by a subject is called her/his current task. A function

CT: S → T

is defined, where $CT(S_i)$ is the current task of subject S_i .

Authorized Tasks AT: AT is a function that defines a non-empty set of tasks for a subject that this subject is authorized to perform.

AT: S → $2^T \setminus \emptyset$ (2^T denotes the set of all subsets of T),

where $AT(S_i)$ is the set of tasks that S_i is authorized to perform.

Purposes P: Every task has to serve a certain purpose. Besides, personal data have to be collected for a certain purpose. Purposes have also to be defined according to the system's applications. Authorized purposes shall be modeled by a set P of purposes:

P = set of purposes = $\{P_1, P_2, \dots\}$.

Purposes and tasks can be hierarchically structured (see [Bräutigam/Höller/Scholz 90, p.47]). Purposes for example could be divided into different subpurposes or combined into (in the hierarchy) higher purposes. The same is true for tasks. Privacy aspects and practical reasons have to be considered when choosing an appropriate level in this hierarchy. The purposes of this level are used to define the elements of P. Consequently, only purposes of this level in the hierarchy and of higher levels can be modeled by elements and non-empty subsets of P.

The elements of T have to be defined according to this hierarchy level. Each task has to serve exactly one purpose, but each purpose can be achieved by the performance of different tasks. Different purposes are achieved by disjunctive sets of tasks.

Purpose function for tasks TP: Each task has to serve exactly one purpose.

A function

TP: T → P

is defined, where $TP(T_i)$ is the purpose of task T_i .

Purpose function for objects OP: Each object has exactly one purpose for which the personal data were collected. As objects can consist of different objects of finer granularity (e.g. a file consists of different records) that in turn serve different purposes, non-empty subsets of P are taken to define the purposes for each object.

A function

OP: O → $2^P \setminus \emptyset$

is defined, where $OP(O_i)$ is the purpose for which the object O_i was obtained. If, for example, the purpose $\{P_1, P_2\}$ is defined for an object, this means that this object serves a (higher) purpose which consists of the subpurposes P_1 and P_2 .

Rights R: The access rights that a subject can have to an object are defined by access attributes $R = \{r, w, e\}$,

where r stands for read, w for write and e for execute.

Necessary accesses NA: For any task, it has to be defined in advance which accesses to which objects are needed to perform this task. This is done by defining the set NA which consists of triples of the form (T_i, O_j, x) .

$(T_i, O_j, x) \in NA$ means that for the performance of task T_i the x -access to object O_j is necessary, $x \in \{r, w, e\}$.

Current access set CA: A current x -access, where $x \in R$, by a subject S_i to an object O_j in the current state is represented by a triple (S_i, O_j, x) . The current access set CA is a set of such triples representing all current accesses.

Consent C: According to most national privacy laws, the processing and use of personal data shall also be admissible, if the data subject has consented. A set C is defined as a set of pairs (P_i, O_j) . The pair (P_i, O_j) means that the data subject has consented that his/her personal data contained in O_j are processed for the purpose P_i .

b.) Invariants (privacy properties)

The following invariants define (necessary, but not sufficient) conditions for a system state to meet specific privacy principles. They formally define the privacy policy stated above. To enforce this privacy policy, it has to be guaranteed that these invariants are fulfilled in each system state that is defined by the state variables.

1. *A subject's current task has to be authorized for the subject (authorization property):*

$$\forall S_i: S : CT(S_i) \in AT(S_i).$$

2. *A subject shall only have current access to an object, if this access is needed to perform the current task (necessity of data processing):*

$$\forall S_i: S, O_j: O : (S_i, O_j, x) \in CA \Rightarrow (CT(S_i), O_j, x) \in NA.$$

3. *A subject shall only have current access to an object, if the purpose of its current task corresponds to the purpose for that the object was obtained or if there is a consent from the data subject (purpose binding):*

$$\forall S_i: S, O_j: O : (S_i, O_j, x) \in CA \Rightarrow \{TP(CT(S_i))\} \subseteq OP(O_j) \vee (TP(CT(S_i)), O_j) \in C.$$

c.) State transition functions:

State transition functions that describe changes of state variables that may take place, have to be defined for actions such as *get access, release access, create object, delete object, change current task*.

Besides, privileged functions are needed to define and change new subjects, tasks, authorized tasks for a subject, necessary accesses, purposes of tasks and consents. These privileged functions shall be executed by the security administrator. But the definitions of these sets and functions should be done in cooperation with another person who cares for the privacy

interests of the data subjects (e.g. representative of the works council, data protection commissioner).

The security administrator is thus responsible for enforcing the privacy policy. The privacy policy is non-discretionary, as users cannot pass access rights on to others users at their discretion. The privacy model shall enforce a form of mandatory control that is not based on multilevel security requirements and is therefore different from MAC as defined in the TCSEC.

5.2 Enforcement

The implementation of such a privacy model on the operating system level has the advantage that control can be implemented on the lowest system level within a security kernel. On the other hand, on operating system level, control is only possible on the granularity-level of files. A privacy policy should also be supported at database or application level where access control is possible on granularity of records or elements and where *purpose binding* and *necessity of access* could be further checked in dependence on the element's values. In order to support this possibility of an additional value dependent control at database or application level, the privacy property of purpose binding only demands that the purpose of the current task has only to be a subpurpose (subset) of the purpose of the object and that it has not necessarily be equal to it. Global control is possible at operating system level, e.g. when a file is opened it can be checked whether the purpose of the current task is part of the purpose of the object. A finer control is then in addition possible at database level, where purpose binding can also be checked dependent on the element's values and where the equality of the purposes can be checked. An interesting approach to implement privacy controls on database level was introduced by [Bräutigam/Höller/Scholz 90].

In contrast to TCSEC, more recent Security Evaluation Criteria such as ITSEC [ITSEC 91], CTCPEC [CTCPEC 93] or the Draft Federal Criteria [FC 92] do not require a particular security policy. A system that enforces the privacy model could be evaluated according to these criteria. The privacy model can be enforced together with other security models. This could be done according to Hilary Hosmer's multipolicy paradigm [Hosmer 92].

5.3 Comparison to other security models

The concept of such a formal privacy model that restricts access of a user based on his/her current task, is similar to the concept of role-based access controls (as introduced in [Ferraiolo/Kuhn 92]) that restrict access of a user based on his/her role that he/she is currently performing. Like the role-based model, the privacy model has the advantage of easy administration of the user rights. But in contrast to the role-based model by Ferraiolo and Kuhn, where roles can be hierarchically structured (roles can be composed of roles) with inheritance of rights, the requirement of necessity of data processing forbids inheritance of rights and therefore requires that only tasks of one hierarchy level are modeled.

The integrity-principles of Well-Formed Transactions and Separation of Duties, that are enforced by the Clark Wilson model [Clark/Wilson 87], can also be realized by the privacy model approach.

Transactions could be introduced to the privacy model by attaching a set of transactions to each task that are needed to complete this task. Consequently, the set NA has to be changed

to include tuples of the form $(T_i, O_j, \text{TRANS}_k, x)$ meaning that for the performance of task T_i the x -access to object O_j by performing transaction TRANS_k is needed. The principle of separation of duties can be achieved, if the sets of authorized tasks that are attached to the subjects and the sets of transactions that can be attached to the tasks are chosen appropriately.

6. A holistic approach towards a privacy-friendly use of security mechanisms

Security mechanisms should not only directly address privacy requirements, but should also be used in a privacy-friendly way to resolve the conflict between security and privacy. As the problem of vulnerability is not only a technical problem, but has also social, legal, psychological dimensions, a holistic approach towards a privacy-friendly use and implementation of security mechanisms is necessary that has to involve specialists from different disciplines. So besides the enforcement of a privacy policy to protect the privacy of the system's data subjects, such a holistic approach is also needed to protect the personal control data about users and uses produced by security mechanisms. This holistic approach should particularly contain the following mechanisms (see [Fischer-Hübner/Yngström/Holvast 92]):

a.) Educational Mechanisms

The persons that are responsible for system security, such as the system-designers, auditors and security administrators, should be taught about the privacy-interests and rights of the users and uses. An important countermeasure to vulnerability could be gained, if information and understanding is given to planners and practitioners of IT-security as well as to users and uses.

b.) Legal Mechanisms

Special legal attention is needed to restrict the use of personal data needed by security mechanisms to only security purposes, to prohibit its misuse and to control its use:

Usage for security purposes only:

The principle of purpose binding that is already a requirement of most western privacy acts must also be applied to personal data collected or processed by security mechanisms. The German Data Protection Act in § 14 IV BDSG therefore restricts the use of personal data collected for the purpose of monitoring data protection, safeguarding data or ensuring proper operation of a data processing system exclusively for such purposes.

The right to be informed:

According to most western privacy acts, data subjects have the right to be provided with information on data concerning them, as well as the right to have incorrect data corrected and illegally stored data deleted. However, it might be forbidden to inform the user or usee about data collected on him/her by security mechanisms, if these data fall into a legal clause for secrecy. Users and usees should at least be informed about the kind of events that are audited.

It is being discussed, whether data subject should not only be informed about the stored data concerning him/her, but also about how their personal data are being processed, as otherwise

they would not have the chance to control the correctness of processed personal data. For intrusion detection expert systems the question is raised, whether rules, that extract personal information out of audit data should be known by the users. Otherwise, they would not have the chance to control the correctness of personal data produced. On the other hand, if so-called a-priori rules that encode information about system vulnerabilities and hacker strategies are known by the users, the rules will not be capable of protecting against such kinds of attacks any longer. Furthermore, these rules can also contain sensitive information about the system's vulnerabilities, e.g. about bugs or virus replication techniques - if rules for dynamic virus detection are used -, that should be kept secret, as otherwise they could be misused. Consequently, users should just have the right to be informed about the kind of rules being used.

The draft "Privacy for Consumers and Workers Act", that was introduced to the U.S. Congress in 1990 and intends to prevent potential abuses of electronic monitoring in the workplace, attempts to regulate how an employee must be informed by their employers about the form of electronic monitoring to be used, personal data to be collected, the use of personal data collected, its interpretation, etc. prior to the monitoring process. It also states that employees shall have access to all personal data obtained by electronic monitoring. This act lacks any restrictions on the monitoring process and on the volume of data, that are allowed to be collected.

Participation of the work council:

According to the German Workers' Legislation ("Betriebsverfassungsgesetz") the workers representation (Works Council, "Betriebsrat") must participate in the decision to introduce any system, that can be used or misused for performance monitoring. Arrangements that were made without the participation of the works council are regarded as invalid. Therefore, security mechanisms that can be misused must be accepted by the works council. If an intrusion detection system is accepted by the works council, they should also have influence over what actions are being monitored and the profiles used. It should be discussed, whether the use of user profiles that store information typically needed for performance control, should be prohibited.

c.) Technical mechanisms

Besides the legislative means to control the collection and use of personal data used by security mechanisms, technical protection of that data are also needed.

Protection from illegal and unnecessary accesses:

Security relevant personal data, such as data in password files, access rights databases, audit trails and in profiles of intrusion detection systems, have to be protected from illegal and unnecessary accesses.

Audit trail data should just be read by an auditor for security relevant analysis only. For intrusion detection systems, further control mechanisms are needed that restrict accesses to the profiles in the knowledge base. The auditor shall only have write-access to the knowledge base to influence the intrusion detection process by defining the statistical profiles or adding new rules. He/she should on principle only gain read-access to the knowledge base in cooperation with the works council or with another person that cares for the privacy interests of the users. Only alarm reports and summary reports should be directly readable by the auditor.

Reaching anonymity through the use of pseudonyms:

Another approach to protect personal access control information or personal information in audit trails or intrusion detection profiles could be the use of pseudonyms instead of real subject-names. Methods for a privacy-friendly design of transaction systems by the use of digital pseudonyms were already introduced by [Chaum 85].

For example, formal anonymity of audit trails or intrusion detection profiles could be achieved, if pseudonyms are used in the audit records and replace the subject identifiers that are used in the system. The replacement function could be implemented by an encryption procedure. Proper key management has to guarantee that the key is kept secret and can only be derived by an auditor in cooperation with the works council in case of a security alarm, that has to be analyzed in more detail (e.g. in order to unmask an intruder).

The question, whether the audit trails or profiles still describe personal data, depends on the question, whether a reidentification of subjects is comparatively easy. This question depends in turn on whether the encryption key can be considered as secret and on the additional knowledge, that could be known and used by a potential attacker for reidentification. Audit trails and profiles contain the data or statistics about the actions and behavior of subjects or subject groups. So information about the typical behavior or actions of subjects could be used for reidentification. But such information is often not known by an attacker and represents information that an attacker wants to get at via reidentification. As more and more of such information can be accumulated over some time, the pseudonyms should be changed regularly after a certain time interval. The problem of reidentification must always be carefully analyzed from case to case.

Such methods for reaching formal anonymity of control data by the use of pseudonyms should be further elaborated.

7. Final Remarks

It was discussed that today's security mechanisms rarely address privacy directly and can be in conflict with privacy interests of users and uses.

The privacy model introduced in this paper should help to develop systems that complement other security properties by adding privacy as a security goal. Of course, this model can merely help to enforce specific privacy requirements, but it is not sufficient to guarantee privacy in general, as other security models can only help to enforce specific security properties but not security in general. For the enforcement of the model's privacy requirements, appropriate administrative measures are also needed.

Finally, it has to be mentioned that the holistic approach to a privacy-friendly use of security mechanisms is only one attempt to cope with the conflict between security and privacy and has to be further refined.

References:

- [*Bell LaPadula 76*] D.E. Bell, L.J.LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", Mitre Cooperation, Bedford, Mass. 01730, Januar 1976.
- [*Bräutigam/Höller/Scholz 90*] L.Bräutigam, H.Höller, R.Scholz, "Datenschutz als Anforderung an die Systemgestaltung", Westdeutscher Verlag, 1990.
- [*Chaum 85*] D. Chaum, "Security without Identification: Transaction Systems to make Big Brother Obsolete; Communications of the ACM 28/10, 1985, p.1030-1044.
- [*Clark/Wilson 87*] D.D. Clark, D.R. Wilson, "A Comparison of Commercial and Military Computer Security Policies", Proceedings of the IEEE Computer Society Symposium on Security and Privacy, Oakland, 1987.
- [*CTCPEC 93*] The Canadian Trusted Computer Product Evaluation Criteria, Canadian System Security Centre, Version 3.0e, January 1993.
- [*Denning 86*] D.Denning, "An Intrusion Detection Model", Proceedings of the 1986 IEEE Computer Society Symposium on Security and Privacy, Oakland, 1986.
- [*Denning et al. 87*] D.Denning, P.Neumann, D.Parker, "Social Aspects of Computer Security", Proceedings of the 10th National Computer Security Conference, Baltimore, 1987.
- [*FC 92*] Federal Criteria for Information Technology Security, Draft Version 1.0, NIST & NSA, December 1992.
- [*Ferraiolo/Kuhn 92*] D.Ferraiolo, R.Kuhn, "Role-Based Access Controls", Proceedings of the 15th National Computer Security Conference, Baltimore MD, October 1992
- [*Fischer-Hübner/Yngström/Holvast 92*] S.Fischer-Hübner, L.Yngström, J.Holvast, "Addressing Vulnerability and Privacy Problems generated by the Use of IT-Security Mechanisms", Proceedings of the IFIP 12th World Computer Congress, Volume II: Education and Society, Madrid, September 1992, Ed.: R.Aiken, North Holland
- [*Hosmer 92*] H.Hosmer, "The Multipolicy Paradigm", Proceedings of the 15th National Computer Security Conference, Baltimore, October 1992.
- [*ITSEC 91*] Information Technology Security Evaluation Criteria (ITSEC), Provisional Harmonised Criteria, June 1991.
- [*Ketelaar/Fischer-Hübner 93*] R.Ketelaar, S.Fischer-Hübner, "On the Cutting Edge between Security and Privacy", Proceedings of the IFIP WG 9.6 Conference 'Security and Control of IT in Society', Stockholm-St.Petersburg, August 1993, Ed.: R.Sizer et al., North Holland.
- [*TCSEC 85*] DoD Trusted Computer Systems Evaluation Criteria, DoD 5200.28-STD, Washington D.C., Department of Defense, 1985.

USING A SEMIFORMAL SECURITY POLICY MODEL 2C A C2 BETTER¹

Marvin Schaefer²
Arca Systems, Inc.
10320 Little Patuxent Pkwy, Suite 1005
Columbia MD 21044

Gary R. Grossman
Jeremy J. Epstein
Cordant, Inc.
11400 Commerce Park Drive
Reston VA 22091

ABSTRACT

Informal security policy models are not required by TCSEC until class B1, and formal models not until class B2. However, they can be useful at lower levels of assurance. This paper describes why we developed a semiformal model for Trusted NetWare, a C2 system. The portion of the model that describes access rights to the NetWare Directory Services (NDS) Directory Information Base (DIB) is shown.

Keywords: Models, access control policies, DAC, Novell NetWare, X.500.

1. Introduction

Security policy modeling has a long history of use as a tool that aids in designing, understanding, and analyzing the functionality and properties of a trusted system and its trusted computing base. Policy modeling is required as part of the assurance evidence for trusted systems that enforce policies appropriate to Divisions B and A of the *Trusted Computer System Evaluation Criteria* (TCSEC [4]) and the *Trusted Network Interpretation* (TNI [5]) and *Trusted Database Management System Interpretation* (TDI [6]) of the TCSEC. Such policies include provisions for both discretionary and nondiscretionary aspects of access control. There is no TCSEC requirement for policy models of systems that only enforce discretionary access control (DAC) policies (i.e., systems appropriate to TCSEC Division C).

The authors are engaged in the analysis and development of a commercial C2 distributed network operating system. This product, Trusted NetWare®, integrates access control over network objects and metaobjects (objects that control or define properties of the network itself), over user objects (e.g., files), and over local workstation objects (e.g., private files on a temporally shared workstation). The resulting composed product has a very rich access control policy, and the authors became convinced at an early stage in the project that a unified security policy model could be of considerable value to understanding and controlling development and evaluation of the system and its future releases.

¹Copyright © 1994 Cordant Inc.

²Email addresses for the authors are: Schaefer: marv@arca.md.com; Grossman: ggross@cordant.com; Epstein: jepstein@cordant.com

The resulting model consists of informal (natural language prose) and formal (functional set theory) descriptions. An algebraic specification style was chosen over the more traditional state transition modeling, as this was found to be more natural to understanding the policy. As a means of assessing consistency, the authors produced a few informal proofs of selected invariant relationships during the model's development. While this analysis was useful, full formality was not needed to obtain benefits from the modeling exercise.

The evolving semi-formal model has already shown itself to be valuable to the effort. Developers have begun to find that mathematical notation makes it a precise and concise reference document. Analysts, including the authors, have used the modeling process to identify deficiencies and security flaws in the initial system conceptualization.

1.1. Overview of the Paper

This paper is intended to show the value of semi-formal access control modeling to improve the design and analysis of C2 products. Although [semi-]formal models are frequently used to describe composed systems that implement mandatory access control policies, we believe our use of such modeling to be unprecedented, as it is not a requirement of the TCSEC or the TNI at the C2 level. However, we have found that the resulting model aided significantly in our ability to understand the complexities and subtleties of the discretionary access control policies of NetWare's NDS and File System, of Cordant's Assure product, and of some of the ways in which these policies compose with each other.

We believe that the community should be aware of the ways in which this modeling effort has improved our understanding of the original existing product, of the target trusted system we analyzed, planned and discussed at the whiteboard, and the resulting trusted system undergoing detailed design refinement at this stage of the project. We are confident that the target system will offer assurances that could not otherwise have been established were it not for the insights that resulted from the detailed analysis of the modeling effort.

This paper establishes that:

- Models are useful at C2 for finding flaws and inconsistencies in design and in concept.
- Models are good for placing concepts in proper perspective and for making a complex policy understandable. This is important, since DAC policies are generally targeted to a specific application and are arguably more complex than MAC policies.
- The application of formalism produces a framework for systematically placing concepts, their properties and interrelationships in perspective. The formalism normally employed by mathematicians establishes whether concepts are well-defined and complete, and forces correspondences to be produced and mapped.
- The utility of a model lies in its power to predict consequences of specific system actions and to produce answers to unanticipated questions from developers.

2. C2 Security Policy Modeling

Almost all recent security policy models have been developed or adapted to address variants of military-style Division A and B policies. In these models, the basis for nondiscretionary access control (or MAC) decisions is based on the interpretation of classification labels associated with objects containing information and clearance labels associated with the system's users and with the active subjects that represent users. The models generally exploit the natural partial orderings of these labels as a primary basis for granting or denying access. Discretionary access controls (DAC), based on the concept of *need-to-know*, are normally relegated to a secondary rôle in these models, and a MAC prohibition necessarily dominates access control permissions based purely on DAC.

The most celebrated of formal MAC models is the family of models stemming from collaborative work of D. Elliott Bell and Leonard J. La Padula [1]. These models are characterized by mediation based on the current state of subjects and objects: state transitions are constrained by the set of labeled objects to which each labeled subject currently has *active* classes of observe and modify access. In these models, DAC permissions are represented conceptually as a complete access control matrix (ACM) [*à la* Lampson [2]], each cell of which specifies the full set of permitted modes of access between a subject and an object.

It is possible to model DAC either as a matrix or as a function. In the former approach, a conceptual matrix M is produced such that there is a row for each subject, s , and a column for each object, o , and M_{so} is the complete set of type-specific modes of access granted to s for o . In the latter (logically equivalent) approach, a total discretionary access rights function ds is defined such that $ds(s,o)$ is the set of type-specific access rights of s to o . The names of the subjects that are processes may also occur as objects, since many DAC models control interaction (e.g., interprocess communication) between subjects. More precisely, the names of subjects as column headers in M represent the object-attribute of subjects (i.e., their address space), so that $M_{ss'}$ represents the permitted access rights of s to the address space of s' .

Note that rights entered into M_{so} may be primitive universal system rights like *read*, *write*, *append*, *execute*, ..., or they may extend to include constrained access to o through specific functions or mechanisms. That is, a type-specific right to an access mode to o like *read* may entail (in the model's interpretation) that o be read by invocation of a specific function f rather than directly. In the absence of explicit rights in M_{so} to any alternative viewing mechanisms, g , this generalization of access models would force all of s 's accesses to o to be performed through f . Such representation could distinguish between those subjects (or users) permitted to access a database directly, and those forced to do so only through a controlled viewing or updating mechanism. Alternative modeling approaches are possible and practical, depending on the goals of the model's consumer. E.g., a user u that does not have the right to access a database D directly would have no access rights entered in the cell M_{oD} but would have *invoke* privilege in the cell M_{of} , the cell M_{fD} would show direct modes of access to D . We prefer the former, as it provides a direct answer to the question "who may access D and how?" while the indirection in the latter renders the answer somewhat more inscrutable.

Some DAC models distinguish between the absence of any rights by s to o and a prohibition of s to a defined set of access modes to o . The former may well be represented by those cases where $M_{so} = \emptyset$, but this could represent the ambiguous situation where either s has neither been granted nor denied rights to o or the situation where s has not been granted explicit rights to o but has had specific rights *denied* to o . The ambiguity can be avoided by defining $M_{so} \equiv \langle R_{so}, \neg R_{so} \rangle$, where R_{so} is the set of rights explicitly assigned to s for o and $\neg R_{so}$ is the specific enumeration of type-specific access modes prohibited for s to o . That is, $R_{so} \equiv \{r_i \mid r_i \text{ is a permitted access mode}\}$ and $\neg R_{so} \equiv \{\neg r_i \mid \neg r_i \text{ is a prohibited access mode}\}$. Note that each of $R_{so} = \emptyset$ and $\neg R_{so} = \emptyset$ is independent of whether or not $M_{so} = \langle \emptyset, \emptyset \rangle$.

The Trusted NetWare system does not implement negative access controls. However, the present model provides a framework for their representation should this become a future objective.

It is well-known that the access control matrix, M , can be used to model either Access Control Lists (ACLs) or Capabilities. That is, the column $M_{_o}$ lists the rights for each subject s to o and is o 's ACL; the row $M_{s_}$ lists all of the objects and the modes of access to them permitted to subject s , and is the equivalent of s 's capability list.

Because of the richness of the Trusted NetWare's tripartite policy and because of the object-oriented design and data structures, we have chosen not to use a single matrix, but instead to use functional notation through a series of effective rights functions $ER(s,o)$ that return the set of type-specific modal rights of s to o . This makes the model less abstract and much closer to the implementation.

2.1. NDS DAC Synopsis

This paper focuses on aspects of the discretionary access control policy for the Trusted NetWare Directory Services (NDS) Distributed Information Base (DIB) [7] with the goal of deriving a closed form determination of the effective rights of users to objects in the DIB. NDS is Novell's implementation of the X.500 global naming standard.

Access control policy mediation and policy enforcement are provided by NTCB components on the NDS, the file servers, and on the client workstations. This paper addresses part of the NDS access control policy.

NDS's tree-structured DIB defines the attributes of classes (these are called 'properties' in object instances) and class relationships of *all* Trusted NetWare Objects in typed nodes. That is, the DIB is a database for the overall system, and contains information about all of the system resources (e.g., printers, devices, mounted data servers) and about all of the users who may access any of the system resources. It is important to recognize that the elements of the DIB are *not* synonymous with the objects they describe. Obtaining access to the contents of a DIB node is not, in general, sufficient to obtain access to the described network object.

DIB Objects are treated as instances of encapsulated abstract data types that can be accessed only through well-defined interfaces. DIB nodes are either *containers* (i.e., they may have descendent nodes) or *leaves* according to their class definition. Class instances inherit most properties from their Superclass Objects as in most object-oriented programming models. Trusted NetWare introduces a powerful generalization on objects' inherited access control properties to other objects. Properties defined or inherited for a class are either *mandatory* or *optional*: a mandatory property must be assigned a non-null value; an optional property may be assigned a null value.

The Trusted NetWare NDS implements a discretionary access control policy. Rights are accorded to various forms of classes of subjects, all of which are classed as *objects* in Trusted NetWare's object-oriented nomenclature. Each object instance contains an attribute, called *Access Control List (ACL)*, associated with it that defines access control rights of other objects to it as an object and to subsets of its properties. The Access Control List always contains inherited values relative to certain properties of the object (determined by the object type) and other values that may be assigned by authorized objects.

Within the DIB, every access request is mediated with the effect that changes in user (or object) access rights are immediate. This includes immediate revocation of access rights.

3. Trusted NetWare Directory Services

This section describes and formalizes definitions and rules for the discretionary access control policy for the Trusted NetWare Directory Services Directory Information Base. We incorporate common notation from algebraic set theory and graph theory.

Trusted NetWare Directory Services (NDS) is the basis for structure in the network system. It is represented in terms of the Directory Information Base (DIB), a singly-rooted tree structure that serves to define much of the effective access rights of the active subject relative to NDS objects. The objects in the Directory represent information about a network resource (e.g., about a user, group, printer, volume, etc.). Each object in the DIB is a member of a named class, and inherits properties from its superclass(es). There is a unique class, *Top*, which is the superclass of all classes, and which has no superclass. All classes inherit specific default attributes from *Top*, including the ACL attribute and the *Mandatory Object Class* attribute. Each object's structure (i.e., its set of attributes and properties) and its placement (i.e., its parent's required object class) in the NDS Directory tree is defined and constrained by its *Object Class* attribute.

The tree structure is represented in the model as a mapping by a successor function Γ of ObjectNodes into the set of ObjectNodes:

$$\Gamma: O \rightarrow 2^O \quad (\text{the DIB node successor function})$$

Note that Γ does *not* map objects from the DIB to any of the file servers' nodes or directory trees, or vice versa. We write Γn as a simplification of $\Gamma(n)$. $\Gamma^{-1}n$, represents the inverse image of n under Γ , and is the set of immediate predecessors (i.e., the parents) of n . In a tree, either $\Gamma^{-1}n = \emptyset$ (i.e., n is the root of the tree) or n has a unique parent: $\#\Gamma^{-1}n = 1$.

In order for the DIB to be a tree, it is necessary that only the root R have no immediate predecessor and that all other nodes have exactly one immediate predecessor node, i.e., the following must hold:

$$\forall n \in DIB, (\Gamma^{-1}n = \emptyset \Leftrightarrow n = R) \wedge (n \neq R \Rightarrow \#\Gamma^{-1}n = 1)$$

The ObjectNodes in the NDS DIB are partially ordered. An ObjectNode is represented as a cartesian product (sextuple) consisting of its Class, Superclass, Containment, Name, a set of Mandatory Attributes, and a set of Optional Attributes.

$$\begin{aligned} O \equiv \text{ObjectNode} &\equiv \text{Class} \otimes \text{SuperClass} \otimes \text{Containment} \otimes \text{NamedBy} \otimes 2^{\text{MandatoryAttributes}} \otimes \\ &2^{\text{OptionalAttributes}} \\ &\equiv c \otimes sc \otimes co \otimes nb \otimes 2^{ma} \otimes 2^{oa} \end{aligned}$$

The DIB consists of three classes of node objects:

- **Root object:** the *unique* node that has no parents in the DIB hierarchy
 $R \equiv \text{RootNode} \equiv \{ \exists | r \in DIB \ni \Gamma^{-1}r = \emptyset \}$

- **Container object:** a node that by class definition can contain other objects, which may be either container objects or leaf objects; e.g., an Organization or an Organizational Unit
- **Leaf object:** a node that by class definition can *not* contain other objects, e.g., a user, group, Trusted NetWare server, printer, or other network resource.

$$L \equiv \text{LeafNodes} \equiv \{ l \in \text{DIB} \ni \Gamma l = \emptyset \wedge \mathcal{A}(l) \}$$

$$\text{CN} \equiv \text{ContainerNodes} \equiv \text{DIB} \cap \sim L$$

If a node d is in the Directory, then there is a unique path $\mu(d, R)$ to d from the root node.
If $d \in \text{DIB}$ then $\exists! \mu(d, R)$

The path from the root of the tree to a particular object forms the object's *complete name* or *distinguished name*. Paths in the DIB and in the File System are defined in NetWare documentation such that the destination node is written to the left of the source node.

$$\mu(n_d, n_s) \equiv \text{path from } n_s \text{ to } n_d \equiv n_d = n_k \cdot n_{k-1} \dots \cdot n_1 \cdot n_0 = n_s \ni n_j \in \Gamma n_{j-1}$$

The CompleteName of a DIB node $n_d \in O$ is $\mu(n_d, R)$; the CompleteName of a node $n_d \in D_V$ in the file system is $\mu(n_d, R_V)$.

In addition to its distinguished name, an object in the DIB can also be represented and referenced by an *alias*. Any object in the DIB can have an alias; if an object has subordinates, its alias will appear to have the same subordinates. However, the alias itself must be a leaf vertex, and have no subordinates.

Each named object in the DIB tree has a type, and is derived directly from a *class template*. The class template defines attributes that must be present in all instances of the object. These include the Class, $\mathcal{C}(n)$, SuperClass, $\mathcal{S}(n) \equiv \mathcal{S}(\mathcal{C}(n))$, Containment (the only classes from which the object node may depend), NamedBy (the node's Common Name), Mandatory Attributes (those for which a value *must* be present in the object instance, such as Object Class and Surname for a User object) and Optional Attributes (those for which a value *may* be present in the object instance, such as group memberships for a User object).

A *trustee* to an object O is a user, group or other object that has been granted some specific set of rights to O . The trustees to an object o_2 are precisely those objects included in o_2 's ACL:

$$\text{tr}(o_1, o_2) \equiv o_1 \text{ is Trustee of } o_2 \Leftrightarrow \text{acl}(o_1, o_2)$$

All objects inherit an ACL attribute from the Root. At the time it is instantiated, an object's ACL contains default trustee assignments that are defined from the class template. These defaults can be selectively overridden at the time the object node is created. The ACL attribute may also be modified afterwards by objects holding (or inheriting) the equivalent of a trustee assignment to the object with at least the Write right to the ACL.

Commonly used object classes in the Directory tree are: Country, Device, Directory Map, Group, Locality, NCP Server, Organization, Organizational Unit, Person, Print Server, Printer, Profile (used to specify a shared login configuration), Queue, Resource, Server, Top, User and Volume.

The Trusted NetWare Directory is an object to which specific access controls are applied to object and property rights of *its* objects (i.e., of its nodes). These rights are distinct from the rights applied to the file system by a Trusted NetWare file server. Hence, access to a DIB object or one of its properties does not necessarily imply access to the Trusted NetWare object it describes.

3.1. User Account

Each user of Trusted NetWare has an account. The User (account) object is a leaf node in the DIB, and contains *attributes* that are used to control security and to define security-relevant privileges of the user's execution environment to a server. User nodes *depend from* Organization or Organizational Unit nodes (i.e., their ancestors in the directory tree are Organization or Organizational Unit nodes). The User object defines such user attributes as: user name and password, one or more groups to which the user may belong, optional home directory, optional trustee assignments, optional security equivalences, mailbox, and login script.

A UserNode is an object node of class User.

$$UN \equiv UserNode \in \left\{ un = \langle c, sc, co, nb, ma, oa \rangle \in O \mid c = User, sc \in \{Top, Person, OrganizationalPerson\}, \right. \\ \left. co \in \{Organization, OrganizationalUnit\}, seq \in oa, gml \in oa, g \in gml \right. \\ \left. \Rightarrow \exists gn = \langle c', sc', co', nb', ma', oa' \rangle \ni g = nb', \right\}$$

where *gml* is the user node's group membership list. The user name is unique over the entire network. The node also contains the password used throughout the network. No means is provided for any user, including the supervisor, to read passwords associated with any user.

Trusted NetWare objects *belong to* the immediate container node from which they depend in the NDS Directory tree and, by transitivity, to its predecessor container nodes. In particular, every User object *belongs to* the Organization object or Organizational Unit object in which it is defined. (Note that *belong to* is a transitive relationship.)

Trusted NetWare offers the ability for an administrator to grant membership in one or more named *groups* to individual users. Every user may be members of one or more groups. Groups may also be members of other groups. Each group object *belongs to* a unique organization or organizational unit, and its individual members may *belong to* different branches of the Directory tree.

A user object, group object, organizational object (or any other DIB object) may be on the ACL of any object defined in the DIB tree or in the file system. In this case, the object on the ACL is called a *trustee* to the DIB object or to the file system object.

The user's ID and the user's group memberships are used as bases for access control mediation. Each unique user ID may belong to any number of defined groups, but to exactly one organizational object.

A user may be assigned a directory that can be used as a private workspace. Specific privileges or limitations may be associated with directories.

Each User node has a login script attribute that contains commands that are executed for the user each time the user logs in. Note that these commands all execute on the user's workstation, since there is *no* mechanism for users to execute commands directly on a server.

A *security equivalence*, discussed below, is an explicit assignment that allows one User to have the same set of trusteeships and rights as each other User or User group contained in the User object's Security Equals List. Every UserNode contains a SecurityEqualsList of all DIB objects to which the user has *explicit* security equivalence.

$$sel(u) \equiv SecurityEqualsList(u) \equiv \{ o \in DIB \mid u \in UN, u = \langle c, sc, co, nb, ma, oa \rangle, seq \in oa, o \in seq \}$$

User accounts may be subjected to specific restrictions that are independent of the discretionary access control policy. Account restrictions are assigned by administrators acting with either SUPERVISOR or other appropriate privilege. The assignable restrictions include: designated physical workstations from which the user may login; time of day, by half-hour period, when the user is authorized to login; the number of simultaneous logins the user is permitted; the number of times the user can login with an expired password; the number of consecutive times the incorrect password can be given before the account is disabled; disk space allocated to the account; the account balance; and expiration date for the account.

3.2. Groups and Group Nodes

A GroupNode is an ObjectNode of class Group. Every GroupNode contains an attribute, *member*, that lists the UserNodes and GroupNodes that belong to the GroupNode.

$$GN \equiv GroupNode \in \{ gn = \langle c, sc, co, nb, ma, oa \rangle \in O \mid c = Group, member \in oa \ni m \in member \Rightarrow m \in UN \cup GN \}$$

Every UserNode contains an attribute, GroupMembership, that lists the GroupNodes of which the User is a member. Every GroupNode contains an attribute, MemberList, that lists the UserNodes and GroupNodes that constitute the Group.

$$GroupMembership(u) \equiv gml(u) \equiv \{ g = \langle c, sc, co, nb, ma, oa \rangle \mid \langle c', sc', co', nb', ma', oa' \rangle \in UN, g \in GN, gml \in oa', \\ nb \in gml \}$$

$$MemberList(g) \equiv member(g) \equiv \{ m \in (UN \cup GN) \mid g = \langle c, sc, co, nb, ma, oa \rangle, member \in oa, m \in member \}$$

The UserNodes and GroupNodes are distinct and membership is represented consistently in the DIB.

$$(UN \cap GN = \emptyset) \wedge \forall u \in UN, \forall g \in GN, \quad (g \in gml(u) \Leftrightarrow u \in member(g))$$

3.3. Security Equivalences

A User object A can be made to have a security equivalence to another object B by the addition of the name of object B to object A 's security equivalence list. The subject that makes the assignment to object A must write-manage A (i.e., have at least the Write right to the ACL property of object A). Through the use of security equivalences, one can pass his own rights to any other user on the network that he manages.

Every user object is security equivalent to those DIB objects enumerated in its *SecurityEquals* property, *seq*, to every group of which it is a member, or to any object in the DIB that contains it. Every DIB object is security equivalent to every DIB object that contains it.

$$se(o_1, o_2) \equiv \Leftrightarrow (o_1 = \langle c, sc, co, nb, ma, oa \rangle \in UN \wedge (o_2 \in seq \vee o_2 \in gml(o_1))) \vee o_2 \in \mu(o_1, R)$$

Some security equivalences are system defaults and cannot be assigned or revoked. All objects are security equivalent to the following objects:

- The [Root] object.
- Each successive Container object on the direct path from the root node to the object node's Container object (i.e., the user node is security equivalent to every one of its ancestor nodes).
- The [Public] trustee.

These default security equivalences cannot be changed, nor can they be viewed with administrative utilities.

$$\forall o \in DIB, (se(o, [Root]) = se(o, R)) \wedge (se(o, [Public]) = se(o, R))$$

Note that security equivalence is *not* an equivalence relation. User objects are made security equivalent to the [Root] and to each Container object directly from the [Root] to the Container object they are in, which allows any object to be a trustee of another object. User objects are also security equivalent to any group of which they are a member (but are *not* transitively made to be security equivalent to the members of that group). This permits the assignment of file rights or rights for accessing a printer to an Organizational Unit, e.g., rather than just to a group or specific users. This also permits administrators to use the DIB's container objects as groups.

The [Public] trustee is a special trustee that can be added to any object (as well as to server volume directories and files). Whatever rights are assigned to the [Public] trustee are granted to any client, even if that client has not been authenticated to Trusted NetWare Directory Services.

When a system is first installed, the [Public] trustee is granted the Browse right at the [Root] of the DIB. This allows all User objects in the tree to get around the tree. If desired, this right can be removed from the [Root] object. The suggested more secure setup would be to grant the Browse right to the root container of the DIB, thereby allowing only authenticated objects to browse the entire tree, provided that the Browse right is not excluded by an Inherited Rights Filter (see below).

3.4. NDS Access Control Lists

All NDS objects have a property called the Access Control List (ACL). The ACL controls type-specific modes of access both to the object and to individual properties of the object. The ACL consists of a set of ACL entries. For both the object and for the object's defined properties, each ACL entry lists the *trustee* (the object having rights to the object or named object attributes) and what those rights are (specific rights assignments).

Each object in the DIB can have one or more entries in the ACL attribute of any object. Each entry in an ACL is a 3-tuple consisting of: a *subject name*, a *protected attribute name*, and a *rights set*. These are further described below.

- **Subject Name:** This is the complete name of a trustee with some right(s) to the object or to some of its properties.
$$SN \equiv \text{SubjectName} \in \{ \text{CompleteName}, [\text{Root}], [\text{Public}], [\text{Creator}], [\text{Self}], [\text{InheritanceMask}] \}$$
- **Protected Attribute:** The name of the attribute to which the rights set applies. It may instead be an identifier such as [Object Rights] or [All Attributes Rights]. If the field is [Object Rights], the access rights apply to the object of which this ACL is an attribute rather than to its protected attributes.

$PAN \equiv ProtectedAttributeName \in \{AttributeName, ACL, Object, [EntryRights], [AllAttributeRights], SMSRights\}$

- **Rights set:** This field, called the *privilege set* in standard Novell documentation, enumerates the set of access rights that have been granted to a subject relative to the Protected Attribute. If [Inheritance Mask] is being specified, it enumerates the set of allowable rights that may be inherited.

The AccessControlListElements (ACLE) is the set of all elements of access control lists corresponding to existing DIB nodes. Each element is a triple consisting of a subject name, a protected attribute name, and a set of rights. (These elements include the inherited rights filters, which are distinguished by the subject name [InheritanceMask].

$$ACLE \equiv AccessControlListElements \equiv \{ acle = \langle sn, pan, ps \rangle \mid \exists m \in O, \exists n = \langle c, sc, co, nb, ma, oa \rangle \in O \ni sn \in SN \cup \mathcal{M}(m), acle \in oa, pan \in PAN, ps \in PS \}$$

The ACL function yields, for any DIB node, the set of AccessControlListElements that make up its ACL. The *acl* function yields, for any subject and any DIB node, the set of AccessControlListElements that are in the node's ACL and for which the given subject is the subject identified in the access control list element.

$$ACL(n) \equiv \{ acle \in ACLE \mid \exists \langle c, sc, co, nb, ma, oa \rangle \in (O \cup D) \ni acle \in oa, n = nb \}$$

$$acl(s, n) \equiv \{ acle = \langle sn, pan, ps \rangle \in ACL(n) \mid s = sn \}$$

All DIB nodes are created with default ACL $dACL(n)$ determined by the node's class. We define the total function $dACL(n) \mathcal{C}(n)$ on all of the DIB classes. For notational convenience, we define $\Delta ACL(n)$ to be $dACL(n) \mathcal{C}(n)$.

There are five ways to assign one object, O_1 access rights to another object O_2 :

1. Object O_1 's rights to O_2 may be specified in one or more entries in O_2 's ACL.
2. Rights can be assigned to [Root] or [Public].
3. Object O_1 can be made security equivalent to another object that has rights to O_2 .
4. A Parent object of O_1 can be assigned rights to O_2 .
5. O_1 can be assigned rights to a parent or ancestor of O_2 . These rights will apply to O_2 unless they are filtered out by an Inherited Rights Filter (described below).

Note that the ACL enumerates those objects (including User objects) that have access to the particular NDS object or its properties, but it does not list the specific accesses the NDS object itself may have to other objects. So, if a User object, u , is being placed in the DIB, then u 's ACL defines the trustees to u and their particular rights to view or modify contents of the User object's definition (e.g., the Login Script for the User object u). If u were a trustee to other objects (or object properties), the ACL attributes of each of *those* objects would list u 's rights to those objects or to their properties.

The ACL is itself an attribute or property of an object, and the ACL can have (as one of its values) a trustee to itself. If the trustee has the Write right to the ACL, that trustee is allowed to modify any of the rights of the object. (This follows since the trustee could otherwise modify the ACL to grant itself that right).

When an object is created, the creator object automatically receives all object and property rights to the newly created object. However, the creating object will not receive any right that is not effective (e.g., because of an Inherited Rights Filter) at the new object's level of the DIB.

Any right r to an object o is a pair consisting of a ProtectedAttributeName $a.\mathcal{C}(o)$ and an element of the RightsSet defined for $a.\mathcal{C}(o)$. The set of rights contained in an *acl* is:

$$rts(acl(s, n)) \equiv \{ \langle pan, rs \rangle \ni acl(s, n) = \langle s, pan, rs \rangle \}$$

The set of trustees is the same as the set of objects that appear on some object's ACL; any object can be a trustee.

$$T \equiv Trustees \equiv \{ o \in O \mid \exists n \in (O \cup D), \exists acle = \langle sn, pan, rs \rangle \in ACL(n) \ni o = sn \}$$

3.5. Access Rights

Access rights are granted to objects and to attributes.

3.5.1. NDS Directory Object Rights

Each object in the Trusted NetWare Directory has rights associated with it. These rights control what objects (i.e., User objects or other active entities in the system) can do *with* or do *to* the specified object (but *not* with its contents). These Directory object rights are:

Browse	The right to see the object in the Directory tree.
Create	The right to create a new object below the specified object in the Directory tree. (This right applies only to container objects.).
Delete	The right to delete the object from the Directory tree. Leaf objects and Empty Container objects are the only objects that can be deleted.
Rename	The right to change the name of the object. This right only applies to Leaf objects.
Supervisor	The right to all object access rights as well as all rights to the object's properties.

It should be noted that Browse rights are not sufficient to view or modify the *contents* of the object, but only to view the object itself.

$OR \equiv NDSObjectRights \in \{ B, C, D, Ren, S \}$

3.5.2. NDS Properties

All objects have properties, each of which can have at least one value. Rights for a given property apply to all of its values. These rights control what users or other entities in the system can do *to* the named property or properties.

These property rights are:

Compare	The right to compare a value to the value of the property. This does not permit direct viewing of the property value.
Read	The right to read the values of the property. If the Read right is given, compare operations are also allowed even if the Compare rights permission is not explicitly granted.
Write	The right to add, remove, or change any values of the property.
Add or delete Self	The right to add or remove itself as a value of the property. The trustee cannot affect any other values of the property.
Supervisor	All rights to the property.

$PR \equiv NDSPropertyRights \in \{ C, R, W, AS, S \}$

Property rights can be assigned either uniformly to *all* properties of the object or individually to specific properties of the object. For example, when a User object is created in the Directory, the User object itself is given the Read right to all properties of that object. In addition, by default the user is also given Read/Write rights to its Login Script property and the Print Job Configuration property; while the User object is not given default Write access to its login restriction properties.

In summary, the rights set for DIB objects consists of object rights and property rights, *i.e.*:

$RS \equiv DIBRightsSet \in \{ 2^{OR}, 2^{PR} \}$

3.6. Inherited Rights

Object and property rights flow from the top of the NDS Directory structure down the tree. When rights flow down the tree, they are known as *inherited rights*. However, the only rights that can be inherited are [Object Rights] and

[All Property Rights]. However, individual property rights, i.e., trustee rights that are explicitly granted to *specific* properties like the ACL, are *not* inherited.

Inheritance for a specific user can be controlled by making multiple assignments appropriate to the context of operation.

3.7. Inherited Rights Filter (IRF)

The Inherited Rights Filter (IRF) is the Trusted NetWare mechanism that limits inherited rights for all users in the Directory. IRFs are available for Directory objects and for [All Property Rights] assignments. The set of Inherited Rights Filters, *IRF*, is that subset of the set of all AccessControlListElements which are distinguished by the subject name [InheritanceMask].

$$IRF \equiv \text{InheritedRightsFilters} \equiv \{acle = \langle s, pan, rs \rangle \in ACLE \mid s = [\text{InheritanceMask}] \} \subseteq ACLE$$

The function *irf* yields the set of all inherited rights filters for a node.

$$irf(n) \equiv \{ irfe = \langle sn, pan, rs \rangle \in ACL(n) \mid sn = [\text{InheritanceMask}] \}$$

In contrast to trustee assignments, IRFs do not grant rights. They exist for the sole purpose of revoking rights. The IRF explicitly enumerates the rights that *may* be inherited and any right *not* enumerated in the IRF is not inherited. The effect is that for every object that does not have a trustee assignment to this object node is to allow only the specified rights to exist.

In the instance where a specific trustee assignment is granted, the object to which the assignment is granted will override the IRF restrictions. This is true for both object and property rights. They do not affect any rights granted at the point where the IRF is installed.

If the Supervisor *property rights* are disallowed, all objects not specifically having the Supervisor property right granted to the object will be disallowed that right from inheritance. The only exception is the Supervisor *object right*. By default, this right grants all property rights and *cannot* be limited by a property IRF. So, if an object inherits the Supervisor *object right* from above, a property IRF has no effect on that object.

If the Supervisor *object right* is disallowed via an object IRF, this disallows inheritance of the Supervisor right from above. Therefore, one can effectively prevent Supervisor access to portions of the Directory by placing object IRFs which disallow the Supervisor object right from being inherited. *As a consequence of this property, it is possible to block Supervisor rights completely from a part of the DIB.* If all objects with Supervisor rights are deleted relative to a specific section of the Directory tree, there will be *no* effective way of managing that section of the tree.

The set of trustees is the same as the set of objects that appear on some object's ACL; any object can be a trustee.

3.8. Effective Rights to an NDS Object

The effective rights $ER(o_1, o_2)$ of a User object or other NDS object o_1 to an NDS object o_2 are derived *every* time an access is requested. The derivation takes account of the following:

- the object's set of explicit trustee assignments to the NDS object;
- trustee assignments inherited from o_1 's container (implicit security equivalents of o_1);
- if the requesting object is a User object, the trustee assignments to any Group object of which the requesting User object is a member (implicit security equivalents of user);
- if the requesting object is a User object, trustee assignments to any object listed in the requesting User object's list of security equivalences;
- trustee assignments inherited from the NDS object's container, that are not filtered out by the NDS object's IRF;
- trustee assignments inherited from the requesting object's container as a trustee of the NDS object's container that are not filtered out by the NDS object's IRF.

The explicit trustee assignments in a node add to those inherited from its parent node, if any. The Supervisor right can be masked for object and property rights. These rules are different from those that apply to the File System. In

particular, the Supervisor right cannot be masked for file system rights. Further, the Supervisor right cannot be filtered out by an IRF for files and directories.

The effective rights of an object, o_1 to an NDS object o_2 are computed as

the union of

{the union of all explicit trustee rights by o_1 and all objects to which o_1 is security equivalent to o_2 }

with

{the union of all inherited rights of o_1 or any object to which o_1 is security equivalent to o_2 }.

That is,

$$r \in ER(o_1, o_2) \Rightarrow \left\{ \begin{array}{l} (r \in rts(acl(o_1, o_2))) \\ \vee (o' \in UN, \exists o' \in DIB \ni se(o_1, o'), r \in rts(acl(o', o_2))) \\ \vee (\exists o \in \mu(o_2, R) \ni r \in ER(o_1, o)) \\ \wedge \forall o' \in \mu(o_2, \Gamma o), r \in irf(o') \end{array} \right\}$$

3.9. Access Rights Acquisition in the NDS

In the DIB, object o_1 has right r to object o_2 if any of the following holds: o_1 is a trustee of o_2 with right r ; o_1 is security equivalent to a trustee of o_2 possessing right r , or o_1 or a security equivalent of o_1 has right r to a container of o_2 that is not filtered out by an IRF.

The object o may not necessarily access a node $n \in DIB$, even if $tr(o, n)$. In order for object o to achieve any form of access to a node $n \in DIB$, the object must have at least Browse object rights to $\Gamma^{-1}n$; i.e., it is necessary that

$$\emptyset \neq (ER(o, \Gamma^{-1}n) \cap \{ \langle \text{Object}, \{B, S\} \rangle, \langle \text{ACL}, \{W, AS, S\} \rangle \})$$

For some property rights, possession of an access right to a property \mathcal{P} of node o_2 by an object o_1 may imply that o_1 has other rights to the property of o_2 . In particular, if we let $\mathcal{A}(s, P(x))$ mean that s has the capability to make $P(x)$ True, then

$$\begin{aligned} (\langle \text{Object}, \{S\} \rangle \in ER(o_1, o_2)) &\Rightarrow \forall \mathcal{P} \in \alpha(o_2): \mathcal{A}(o_1, \{ \langle \mathcal{P}, \{B, C, D, Ren, S\} \rangle \in ER(o_1, o_2) \}) \\ (\langle \langle \mathcal{P}, \{S\} \rangle \rangle \in ER(o_1, o_2)) &\Rightarrow \mathcal{A}(o_1, \{ \langle \mathcal{P}, \{C, R, W, AS, S\} \rangle \in ER(o_1, o_2) \}) \\ (\langle \langle \mathcal{P}, \{W\} \rangle \rangle \in ER(o_1, o_2)) &\Rightarrow \mathcal{A}(o_1, \{ \langle \mathcal{P}, \{R\} \rangle \in ER(o_1, o_2) \}) \\ (\langle \langle \mathcal{P}, \{AS\} \rangle \rangle \in ER(o_1, o_2)) &\Rightarrow \mathcal{A}(o_1, \{ \langle \mathcal{P}, \{R, W|_{\mathcal{M}(o_1)} \rangle \rangle \in ER(o_1, o_2) \}) \end{aligned}$$

It is easy to prove the following theorem:

Theorem: Possession of Supervisory Rights to an object is equivalent to possession of Write access to the ACL of the object in the sense that:

$$\begin{aligned} \forall o_1, o_2 \in DIB: \\ (\langle \langle \text{ACL}, \{W\} \rangle \rangle \in ER(o_1, o_2)) &\Rightarrow \mathcal{A}(o_1, \{ \langle \text{Object}, \{S\} \rangle \in ER(o_1, o_2) \}) \\ \wedge \\ (\langle \langle \text{Object}, \{S\} \rangle \rangle \in ER(o_1, o_2)) &\Rightarrow \mathcal{A}(o_1, \{ \langle \text{ACL}, \{W\} \rangle \in ER(o_1, o_2) \}) \end{aligned}$$

4. Conclusions

Trusted NetWare has a rich access control policy that derives from the evolution of the mature products from which it is being built: Novell's NetWare and Cordant's Assure. Although it was not required for C2, we found it useful, indeed invaluable, to develop and use an informal security policy model in order better to understand the composed policy of Trusted NetWare's three network TCB components.

In this paper, we have given an example of the derivation of effective rights to objects in the Directory Information Base that serves as the trusted product's metadatabase. We have shown a few productions and theorems that follow easily from the selected modeling notation. While their statement and proofs appear obvious in the present formulation, considerable examination of documentation, commented code, uncommented code — and ultimately, interrogation of developers — was needed to lay the foundation from which to produce these results.

Since writing the preliminary model, potential flaws and omissions in the evolving NTCB design have been identified and resolved, while new ideas have been tested against the model as part of the ongoing development and assurance process.

Inspiration for writing a model for a C2 system came from [3], where a semiformal model was used for a B1 UNIX system and [8] where a formal model for the UNIX setuid mechanism is described.

5. Acknowledgments

We greatly appreciate the assistance of Lawrence Kpodo (Cordant) and Doug Hale (Novell) who helped explain many of the finer points of the NDS security policy.

6. References

- [1] David Bell and Leonard La Padula, *Secure Computer Systems: Unified Exposition and Multics Interpretation*, The MITRE Corporation, July 1985.
- [2] Butler W. Lampson, "Protection", *Proceedings of the Fifth Annual Princeton Conference*, Princeton University, March 1971.
- [3] *B1st Informal Security Policy Model*, Addamax Corporation, July 1989, Document #288-116-B/1.0.
- [4] *Department of Defense Trusted Computer System Evaluation Criteria*, National Computer Security Center, December 1985.
- [5] *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, Version 1*, National Computer Security Center, July, 1987.
- [6] *Trusted Database Interpretation of the Trusted Computer System Evaluation Criteria, Version 1*, National Computer Security Center, April, 1991.
- [7] *NetWare Application Notes*, Volume 4, Number 4, Novell Inc., Part Number #164-000032-004, April 1993.
- [8] Tim Levin, Steven Padilla, Cynthia Irvine, "A Formal Model for Unix Setuid", *Proceedings of the 1989 IEEE Conference on Research in Security and Privacy*, May 1989.

A TAXONOMY FOR SECURITY STANDARDS

Wayne A. Jansen
National Institute of Standards and Technology
A-216 Technology Building
Gaithersburg, MD 20899

Abstract: This paper presents a taxonomy of security standards developed to ensure a systematic review of security standardization areas appropriate to the Department of Defense (DoD) Goal Security Architecture. The taxonomy relies on a simple paradigm based on the notions of uniformity and quality. This approach attempts to provide full coverage of all relevant security standards, yet be simple to understand and apply. The taxonomy is also open to further refinements and adjustments. Because of the flexibility and simplicity of the taxonomy, other initiatives involving the classification of standards may benefit from its use.

1. INTRODUCTION

The DoD Goal Security Architecture (DGSA) [1] presents a comprehensive view on the architecture of an information system from the perspective of security. The orientation of the DGSA is toward future information systems with a focus on both users and information. It emphasizes distinct autonomous information domains that are distributed among networked computer systems. This perspective differs significantly from past treatments of information systems security and necessitates reexamination of the security standards landscape from this viewpoint.

Standards can be broadly defined as "something taken for a basis of comparison, or that which is accepted for current use through authority, custom, or general consent" [2]. Several divisions of security related standards for information technology can be distinguished [3]: those in which security is the primary concern, those in which security is an important but secondary concern, and those in which security is not specifically addressed but are supportive of security. The paper focuses primarily on the first two divisions and assumes the reader has some familiarity with security standards.

There are many ways to view and classify existing and emerging security standards for information technology. Despite the scheme used, the classification of some standards is always difficult or arbitrary. Although no one scheme is perfect, each serves to help interpret the standardization landscape. Here, two principal classes of standards [2] are differentiated:

- (a) Standards for uniformity, and
- (b) Standards of quality.

It is the sameness of a product that is at issue for standards for uniformity, while classification or grading of a product is at issue for standards of quality. With regard to security, these classes of standards match nicely with functionality and assurance considerations. Since matters of assurance are often overshadowed by functionality, achieving a balance between the

two aspects early in the scheme is important. Table 1 contains the initial categories of a taxonomy for security standards based on the uniformity/quality paradigm. The remainder of the paper examines the taxonomy in detail.

Table 1: Taxonomy of Open System Security Standards

Class	Form
Standards for Uniformity	Standards for Interchangeability of People with Equipment Standards for Interchangeability between Products
Standards of Quality	Interpretive Standards Standard Assessment Procedures

2. STANDARDS FOR UNIFORMITY

For the information technology area, the primary motivation for standards for uniformity is that of interchangeability. Two forms of interchangeability standards can be identified:

- (a) Standards for the interchangeability of people with equipment, and
- (b) Standards for the interchangeability between products.

The goal of the former is to capitalize on the knowledge and experience people acquire in a technology area, possibly through the use of one manufacturer's product. The goal of the latter is twofold: to allow products from different manufacturers to work together, and to be able to substitute one manufacturer's product for another's.

2.1 Interchangeability of People with Equipment

Open system standards for interchangeability of people with equipment are sometimes referred to as either driveability or user portability standards. Driveability refers to the notion of look-and-feel of the user interface. User portability is a bit more general and refers to the ability of a user to move among information systems (i.e., be ported from one system to another) without loss of ability or need to retrain. The standardization areas typically associated with people/equipment interchangeability include specifications of the following:

- (a) User interface, and
- (b) Language syntax and semantics.

The undertaking by the Institute of Electrical and Electronics Engineers (IEEE) Portable Operating System Interface (POSIX) committee to standardize the look-and-feel of a graphical user interface [4] is one of the few examples of the user interface standardization area. The specification aims at consistency of behavior between the window systems used by different products. The standard alphanumeric keyboard layout [5] derived from manual typewriters is another example of a user interface standard. Closer to the subject of security, one can envision a set of standard user interfaces for various types of biometric authentication devices.

The language syntax and semantics standardization area covers wide ground. Included are the command line interfaces for shells and utilities, and textual languages for programming, data definition, and data manipulation. The IEEE POSIX user portability extension for shell services and application utilities [6] is an example of the former, while the Sequential Query Language (SQL) database language standard [20] is an example of the latter.

2.2 Interchangeability between Products

Product interchangeability constrains products from different manufacturers to be identical in certain important characteristics. For this to occur, the benefits of expanding the market must outweigh the potential costs of increased competition. Moreover, some characteristics desired by customers may need to remain inexplicit to allow product differentiation.

There are a variety of terms used for product interchangeability with respect to information systems. They include application portability, plug compatibility, device compatibility, and interoperability. Application portability refers to the movement of an application program between information systems (i.e., be ported from one system to another) without loss of ability or need for reprogramming. Application portability depends on access to the underlying functionality of the platform through well-defined programming interfaces and standardized programming languages.

Both plug compatibility and device compatibility refer to the ability to substitute one piece of equipment for another. As with application portability, plug and device compatibility depend on detailed interface specifications. Here, however, the interfaces for equipment are usually specifications of physical connectors and electrical characteristics. Interoperability is akin to compatibility, but conceptually at a system level rather than at a component or device level. It refers to the ability of distributed information systems to exchange information and use it meaningfully. More often than not, interoperability depends on a set of standards rather than on a single standard. Interoperability, besides compatibility of physical interface characteristics, requires compatibility of logical procedures, syntax, and other characteristics concerning communications across the physical interface.

Interchangeability may retard innovation since it implies a degree of interdependency between products. In spite of this shortcoming, product interchangeability standards are perhaps the most active category of standardization since they tend to foster development of broad markets. Standardization areas for interchangeability between products include the following:

- (a) Protocols,
- (b) Security information objects,
- (c) Elements of management information,
- (d) Programming interfaces,
- (e) Information exchange structures, and
- (f) Techniques.

The protocols standardization area is intended exclusively for those standards providing interoperability of distributed information systems. All Open System Interconnection (OSI)

communications protocols are included in this area as well as those protocols associated with the Internet Protocol Suite (IPS).

Security information objects, distinguished as a separate standardization area, are an important subset of the protocols standardization area. Security information objects are elements of security related information conveyed by communications protocols, but defined independently of them. Most security information objects occur at the application layer of the OSI reference model [9] and are defined using an abstract notation (e.g., ASN.1 [21]). A common security label specification is an example of a standard intended for this standardization area.

Elements of management information (EMI) is a standardization area for object class definitions. An object class is an abstraction for the computational resources of an information system. Object classes define the characteristics, controls, operations, notifications and behavior for a category of resource. An instance of an object class corresponds to a specific resource and is called a managed object. Standards in this area include OSI generic definitions of management information [22]. Likewise, all the management information base definitions specified by the Internet community also fall into this area.

There are numerous standards associated with the programming interfaces standardization area. Many of them are interface standards for application programming. An application programming interface defines the functional boundary between application software and the underlying services of the software/hardware platform. Application programming interfaces include such areas as database manipulation, system utilities, and window operations. Programming interfaces may be generic (i.e., language independent) or bound to a specific programming language. The standardization area includes both types of programming interfaces, as well as lower level programming interfaces between platform components. The Generic Security Services Application Programming Interface (GSSAPI) [23] being defined by the Internet community is a good example of a standard in this area.

The standardization area of information exchange structures contains standards that govern the interchange of information between applications on a single or different platforms. Information types include structured data representing documents, graphics, voice, images, et cetera. A specification of a common format audit data is an example standard of this area. Other examples are the Open Document Architecture (ODA) [24] and Standard Generalized Markup Language (SGML) [25] sets of standards.

The techniques standardization area refers to those standards that specify security related mechanisms and algorithms. Standard techniques provide the fundamental building blocks for an implementation, and include standards for data confidentiality, data integrity, digital signature, and key management.

2.3 Synopsis

The standardization areas identified for uniformity standards are summarized in Table 2 below. Note that the set of standardization areas is open to further additions and refinements, as they are identified.

Table 2: Standardization Areas for Standards for Uniformity

Form	Standardization Area
Standards for Interchangeability of People with Equipment	User Interface Language Syntax and Semantics
Standards for Interchangeability between Products	Protocols Security Information Objects Elements of Management Information Programming Interfaces Information Exchange Structures Techniques

3. STANDARDS OF QUALITY

Quality standards provide a means to differentiate between products by either establishing two or more comparative categories or by simply establishing a minimal level of acceptance. Quality standards often involve labeling or registration of products. They are, therefore, more likely than interchangeability standards to require some means of enforcement. Note that, although not their primary purpose, quality standards do allow interchangeability and some interchangeability standards provide minimum quality levels.

The principal function of quality standards is to supply information, and many quality standards are simply agreements on definitions. Quality standards also include standards of identity, design, and performance. Two forms of quality standards can be identified:

- (a) Interpretive standards, and
- (b) Standard assessment procedures.

Interpretive standards are concerned with the acceptability of products and practices, and include definitions of terminology and measurement. For example, specifying fundamental characteristics of an information system, such as throughput, and formulas for calculating related measures apply here. Standard assessment procedures are concerned with the assessability of important characteristics of products and systems. Standard assessment procedures are closely tied to measurement and useful when no single method of measurement clearly stands out. They provide an objective means of obtaining impartial results that are repeatable and reproducible at qualified evaluation laboratories. With some standards, the distinction between the two forms is blurred, making classification difficult.

A pair of standards from the European Commission provide a good example of complementary and distinct interpretive standards and standard assessment procedures. The Information Technology Security Evaluation Criteria (ITSEC) [26] defines a set of functionality and assurance traits, and requirements for assessment of the security capabilities of information technology products and systems. The Information Technology Security Evaluation Manual

(ITSEM) [27] describes how information technology products and systems will be evaluated according to the ITSEC. It contains a comprehensive explanation of the evaluation process, philosophy and principles, and methodology intended to facilitate mutual recognition of evaluation results between European Union member nations.

3.1 Interpretive Standards

For the field of information technology, areas for interpretive standards include the following:

- (a) Meta-standards,
- (b) Terminology,
- (c) Measures,
- (d) Guidance,
- (e) Profiles,
- (f) Methods, and
- (g) Registration.

Meta-standards are standards that govern the content of other standards. A number of examples involve the OSI security architecture [8], an internationally standardized architecture for communications security specified by the International Organization for Standardization (ISO). The security architecture builds upon the OSI reference model [9], which is also part of the standard. The OSI security architecture defines a set of security services and mechanisms and defines fundamental principles to be followed in developing communications protocol standards. The OSI security services include confidentiality, integrity, access control, authentication, and non-repudiation. ISO Security Frameworks [10] extend each OSI security service beyond communications to a broader systems perspective, and add security audit and key management areas. Non-OSI examples of a meta-standard include volume one of the Federal Criteria for Information Security Technology [7], and the ITSEC [26]. Meta-standards generally affect sets of standards categorized in one of the uniformity standardization areas.

Terminology and measures standardization areas respectively contain standards of agreed definitions for common terms and measurements regarding information technology. The guidance area contains standards that give insight to the application and use of other standards. The National Computer Security Center (NCSC) rainbow series contains many documents appropriate for the guidance standardization area. Standards in terminology, measures, and guidance areas are generally regarded as end-products; otherwise, they would belong to the meta-standards area.

The profiles standardization area refers to standards that concisely characterize the traits and capabilities of a functional area. Standards in this area generally reference one or more basic standards, and are oriented toward procurement. There are many varieties of profiles including management ensembles, OSI profiles, security profiles, and protection profiles.

Ensembles are standards for common, yet independent, management tasks. A management ensemble defines how information systems interoperate to solve a specific management problem. An ensemble includes a description of the management context, a definition of the information model, and a scenario description. The management context

provides a view of management capabilities by indicating the level of detail at which resources can be managed. The information model indicates the resources to be managed and the command and control specifications on those resources. The scenario description illustrates how the ensemble management function can be applied to the information model and what activities occur across the communications interface. The need for management ensembles was first recognized by the Network Management Forum and subsequently adopted within regional open systems workshops.

The OSI profiles are functional specifications of layered communications protocol standards that are concerned with pan-layer issues. They provide implementation specific details concerning subsets, options, and parameters of base protocol standards. Many OSI profiles have already been agreed upon within regional implementors' workshops and some of them are being standardized as International Standard Profiles. The Government Open Systems Interconnection Profile (GOSIP) [28] is an example of an OSI procurement profile standard for this area.

A security profile [3] defines the requirements for a common set of communications security services and associated protocols across all seven OSI layers. A security profile is somewhat similar to a Federal Criteria protection profile [7] insofar as it is intended for a large number of applications and contains a description of the target environment, identification of the range of threats to counter, a specification of how security functions counter the assumed threats, a specification of the security mechanisms needed to provide the security functions, and the range of the realizable quality attainable. However, a security profile concerns only communications security and contains extensive details about the protocol mechanisms, while a protection profile generally applies to a broader range of information security technology at a somewhat higher level of abstraction.

Methods are standardized procedures for obtaining a desired result. This standardization area contains cookbook-like standards. One example of such a standard is the NCSC document, Guidance for Applying the DoD Trusted Computer System Evaluation Criteria (TCSEC) [17,18]. Standards for conducting risk analysis also would apply here.

The registration standardization area is a subset of the methods area. It is identified as a separate standardization area due to its important role in rendering reputable catalogs of information. The registration area concerns standardized procedures for the establishment and maintenance of a collection of items. Registration is necessary for many standards to be useful. For example, a registered identity is needed to negotiate a common procedure, algorithm, data format, and other security attributes for communications. Registration also may involve some form of vetting for an item being registered. Most standards in the profiles area require registration, as do security information objects and elements of management information. ISO procedures already exist for the registration of encipherment algorithms [19].

3.2 Standard Assessment Procedures

Standardization areas for assessment procedures include the following:

- (a) Methodology, and
- (b) Reference materials.

The methodology standardization area includes those standards that specify common procedures for testing or evaluating important characteristics of an information technology product. Because evaluation of a product can be a somewhat arbitrary procedure, standard methodologies are needed to provide an objective means for obtaining impartial results or verdicts that are repeatable and reproducible by other assessment facilities. The ITSEM [27], mentioned earlier, is an example of a standard in this area.

The reference materials area refers to those standards that contain standardized test scenarios or other materials for use in product evaluation, classification, or grading. Reference materials are needed for performing such activities as product validation, conformance assessment, and interoperability assessment. Often, there is a close relationship between standards in the methodology and reference material areas. For example, the OSI Conformance Test Methodology [29] and associated test scenarios for various communications protocols respectively provide an example of standards for these areas.

3.3 Synopsis

The standardization areas identified for quality standards are summarized in Table 3. As with uniformity standards, the set of standardization areas listed below is open to enhancement.

Table 3: Standardization Areas for Standards of Quality

Form	Standardization Area
Interpretive Standards	Meta-standards Terminology Measures Guidance Methods Profiles Registration
Standard Assessment Procedures	Methodology Reference Materials

4. SUMMARY

This paper presents a taxonomy for security standards based on the notions of uniformity and quality. The approach taken attempts to provide full coverage of standards in which security is either a primary concern or an important, but secondary, concern. While the taxonomy is at a high conceptual level, many specific examples are given to help understand the scheme and classify other standards. The taxonomy provides a good balance between matters of assurance and functionality, and has proved useful in determining areas requiring standardization with regard to a specific architectural framework for security.

REFERENCES

- [1] Department of Defense (DoD) Goal Security Architecture, version 1.0, Defense Information System Security Program, August 1993.
- [2] David Hemenway, Industrywide Voluntary Product Standards, Ballinger Publishing Company, 1975.
- [3] Taxonomy of Security Standardization, Version 2.0, ITAEGV N69, April 1992.
- [4] IEEE P1201.2, Recommended Practice for Graphical User Interface Driveability.
- [5] ISO 2530, Keyboard for International Information Processing Interchanging using the ISO 7-bit Coded Character Set - Alphanumeric Area, 1975.
- [6] IEEE P1003.2a, Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities.
- [7] Federal Criteria for Information Technology Security, Version 1.0, NIST & NSA, December 1992.
- [8] ISO 7498-2, Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture, February 1989.
- [9] ISO 7498, Information Processing Systems - Open Systems Interconnection - Basic Reference Model, 1984.
- [10] ISO 10181, Security Frameworks for Open Systems, Parts 1-8, June 1994.
- [11] ISO 10021, Information Processing Systems - Text Communications - Message Oriented Test Interchange System (MOTIS), Parts 1-7, 1990.
- [12] ISO 10164, Information Technology - Open Systems Interconnection - Systems Management, Parts 1-10, December 1992.
- [13] Security Association Management Protocol (SAMP), ISP-421, draft revision 0.4, National Security Agency, November 1992, FOUO.
- [14] IEEE P1003.6, Security Interface Standards for POSIX, November 1992.
- [15] ISO 10736, Open Systems Interconnection - Transport Layer Security Protocol, December 1992.
- [16] ISO 11577, Open Systems Interconnection - Network Layer Security Protocol, November 1992.

- [17] DoD 5200.26 - STD, DoD Trusted Computer System Evaluation Criteria (TCSEC), December 1985.
- [18] CSC-STD-004-85, Technical Rational Behind CSC-STD-003-85: Computer Security Requirements -- Guidance for Applying the DoD TCSEC in Specific Environments, NCSC, June 1985.
- [19] ISO 9979, Procedures for the Registration of Encipherment Algorithms, Final DIS Text, June 1990.
- [20] ANSI X3.135, American National Standard for Information Systems - Database Language - SQL, American National Standards Institute (ANSI), October 1992.
- [21] ISO 8824, Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).
- [22] ISO 10165-2, Information Technology - Open Systems Interconnection - Structure of Management Information - Part 2: Definition of Management Information, August 1991.
- [23] J. Linn, Generic Security Service Application Program Interface, Request for Comments (RFC): 1508, September 1993.
- [24] ISO 8613, Open Document Architecture/Open Document Interchange Format (ODA/ODIF), Parts 1-10, 1989.
- [25] ISO 8879, Standard Generalized Markup Language (SGML).
- [26] Information Technology Security Evaluation Criteria (ITSEC), Version 1.2, Commission of the European Communities, June 1991.
- [27] Information Technology Security Evaluation Manual (ITSEM), Draft V0.2, Commission of the European Communities, April 1992.
- [28] Government Open Systems Interconnection Profile (GOSIP), Federal Information Processing Standard (FIPS) 146-1, National Technical Information Service, April 1991.
- [29] ISO 9646, Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework, Parts 1-7, March 1991.

**The Graphical Display of A Domain Model of Information Systems Security (INFOSEC)
Through Semantic Networks: A Description of the INFOSEC Semantic Network for
Information Systems Security Engineers.**

**Teresa T. Smith & Kathleen V. Dolan
National Security Agency
9800 Savage Road
Ft. Meade, Md 20755
(410) 684 7374**

Abstract:

If information is truly our most precious corporate resource, then the efficient and effective retrieval of that information is vital to our success. Equally important is the ability to reuse and record decisions made regarding the design of secure systems. Model-based engineering represents a paradigm shift which provides the capability to structure information and facilitates the reuse of architectures, designs, specifications, and code. Domain analysis is a necessary step in this new paradigm. Domain analysis has been defined as "the process of identifying and organizing knowledge about some class of problems – the problem domain – to support the description and solution of those problems". [2] Our implementation of the model-based engineering paradigm has developed, through domain analysis, a structure for the organization of information from the Information System Security (INFOSEC) domain.

In previous work, a multitude of taxonomies or classes of the information in the INFOSEC domain have been produced. [4,6,11] Although these taxonomies provide a way of categorizing information in the INFOSEC domain, aids that conform to those taxonomies and provide INFOSEC engineers (ISSEs) with access to that information electronically have not existed, until now. Electronic access is important mostly because the complex structure of the relationships within the domain is difficult, if not impossible, to adequately describe in paper format. Furthermore, ISSEs appear to prefer electronic access to information. Currently, in an effort to provide access for ISSEs to information, documents and diagrams are being loaded onto our computer networks, usually without a taxonomic guideline. Therefore, lacking a taxonomic guideline, once this electronic information is available, the situation now changes to a problem of selection, or finding the information that is needed, in the large electronic information base that has been created.

In the Office of Systems Engineering, we have developed an automated tool to provide ISSEs with the ability to efficiently and effectively locate electronic information within the INFOSEC domain. This tool, the INFOSEC Semantic Network, displays a taxonomy of INFOSEC by graphically representing knowledge in the form of a semantic network. The first section of this paper briefly describes domain analysis and the use of semantic networks for domain analysis and information retrieval purposes. The second section presents an overview of INFOSEC Semantic Network taxonomy and knowledge base of INFOSEC, a brief history of the INFOSEC Semantic Network development, and the current capabilities of the tool. Finally, the third section addresses future plans for user testing and potential research directions for the INFOSEC Semantic Network.

Section 1: Domain Analysis and Semantic Networks

If information is truly our most precious corporate resource, then the efficient and effective retrieval of that information is vital to our success. Equally important is the ability to reuse and record decisions made regarding the design of secure systems. Model-based engineering represents a paradigm shift which provides the capability to structure information and facilitates the reuse of architectures, designs, specifications, and code. Domain analysis is a necessary step in this new paradigm. Domain analysis has been defined as "the process of identifying and organizing knowledge about some class of problems – the problem domain – to support the description and solution of those problems." [2] Domain analysis should produce a taxonomic guide to a body of information that is important to the domain. Our implementation of the model-based engineering paradigm has developed a structure for the Information System Security (INFOSEC) domain. In previous work, a multitude of taxonomies or classes of the INFOSEC domain have been produced. [4, 6, 11] The Unified INFOSEC Criteria (UIC) is another project which resulted in the capture of a large base of INFOSEC information. Although the taxonomies provide a way of categorizing information in the INFOSEC domain and other projects have produced large knowledge bases, aids that conform to those taxonomies and provide ISSEs with access to these knowledge bases electronically have not existed.

One method to provide this service is by representing knowledge through the display of semantic networks. Research in semantic networks began with a doctoral dissertation in the 1960's on the use of semantic networks to represent concepts underlying English words. [3] A semantic network displays information in a graphical format. The nodes in the graph are concepts and the links between nodes represent relationships between the concepts. This form of knowledge representation models one theory for human memory; the theory of associative memory. This refers to the belief that our memories are structures of nodes and links in which associative connections are made. This theory has a long history, in fact, Anderson and Bower have traced the idea of associative memory back to Aristotle's time. [1] If we accept the above premise, that humans internally store information in structures similar to semantic networks, than storing information electronically in this manner to provide for similar retrieval should have positive results. The theoretical foundation of this hypothesis is the theory that if we present information to users in a manner in which they are comfortable, i.e. semantic networks, then the users will be able to retrieve information in a more efficient and effective manner. [5] Based on this theory, we have hypothesized that the INFOSEC Semantic Network will aid ISSEs in performing their job duties by providing a method of efficient and effective retrieval of information.

As an example of a semantic network, consider the taxonomy of living animals. The Webster Ninth New Collegiate Dictionary defines the mammal as, "a class of higher vertebrates comprising man and all other animals that nourish their young with milk secreted by mammary glands and have skin usually more or less covered with hair." As the definition states, a human is in the mammal class. Therefore the relationship between the mammal node and the human node is drawn in Figure 1. Another animal in the mammal class is an elephant. This is also depicted in Figure 1. The attributes of the Mammal node are: 1) higher vertebrates animals, 2) nourish their young with milk secreted by mammary glands, and 3) have skin usually covered with hair. Since an elephant and a human are both mammals, they inherit all of the attributes of a mammal automatically. Therefore, in creating semantic networks, it is not necessary to repeat attributes but it is possible to refine them. If the INFOSEC Semantic Network supported Figure 1, the information about humans, elephants, or mammals could be retrieved by a user by clicking on the concept node. This information could be in a textual, video, still pictures, or audio format. The relationship between mammals, humans,

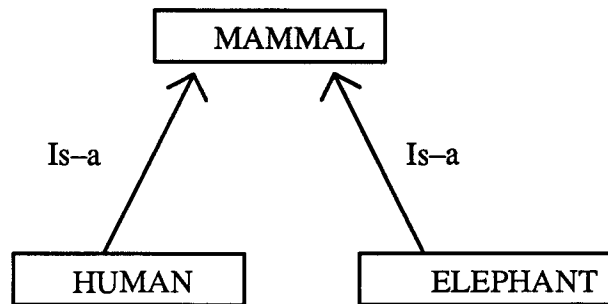


FIGURE 1: GENERIC SEMANTIC NETWORK EXAMPLE

and elephants is another piece of information captured by the semantic network and provided to the user. This relationship information, in some applications, may be as important to display as the information itself.

Section 2: The INFOSEC Semantic Network

This section presents an overview of the INFOSEC Semantic Network taxonomy and knowledge base of INFOSEC, a brief history of the INFOSEC Semantic Network development, and the current capabilities of the tool. The original taxonomy resulted from an effort to evaluate INFOSEC from a top down approach. The top down evaluation resulted in a beginning model that subdivided information system security into two classes: security services and security mechanisms. [8] This was based on the realization that in the process of designing a secure system, the importance of security services is decided. Once security services have been selected, security mechanisms are chosen to meet the required functionality of the security services. Therefore, the first model focused on security services and security mechanisms and their interrelationships. The rationale behind this model was verified by two other sources that began with the breakdown of INFOSEC into security services and security mechanisms. [7, 9]

The basic model began with four basic security services: Integrity, Availability, Authenticity, and Confidentiality. These security services were derived from an evaluation of three sources: ISO 7498 part 2, The Top Down Decomposition Description Paper, and The Taxonomy of Threats and Security Services for Information Systems. [9, 8, 7] Security services were then partitioned into subservices, as shown in Figure 2. The concept of security mechanisms subdivided into a listing of mechanisms that were also discussed in the above references. A knowledge base of approximately 80 mechanisms, (e.g., encryption, access control, object reuse) was created to provide information for the ISSE along with the relationships between these mechanisms and the security services that they support. For example, suppose one declares that data confidentiality is a subservice which has a relationship with the encryption security mechanism. The relationship can be captured by the semantic network and defined as: in the past we have seen data confidentiality provided through the use of encryption at some confidence level.

The basic concepts of security mechanisms were enriched by attaching through the INFOSEC Semantic Network the ability to obtain information from the UIC. The UIC knowledge base covered many concepts considered security services and security mechanisms in our taxonomy. Therefore,

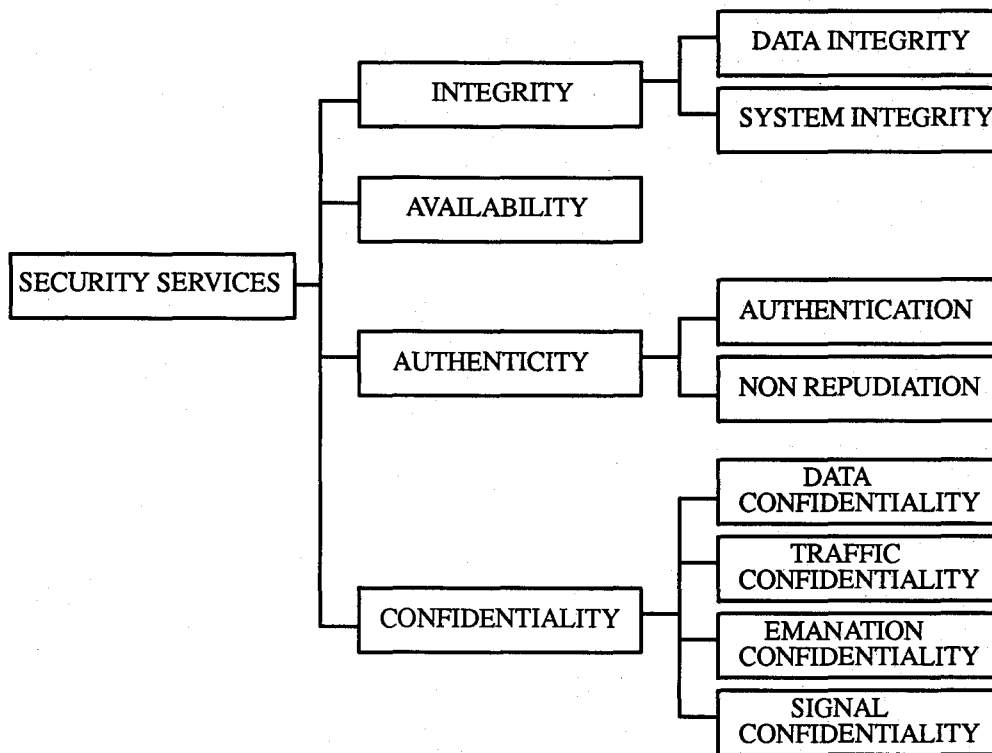


FIGURE 2: SECURITY SERVICES AND SUBSERVICES

it was reasonable to provide the UIC information through the INFOSEC Semantic Network interface. Chapters from the UIC which discussed mechanisms like Discretionary Access Control (DAC) were provided as information and attached to the DAC concept, in Framemaker and Postscript format, in the INFOSEC Semantic Network. Through this attachment, the user only needs to click on the concept and select the background information they wish to read. We also modelled the discretionary access control UIC chapter in our semantic network, breaking it into its' component parts: rationale, requirements, interdependencies, and implementation. The textual information from the UIC gives the user access to approximately 140 written pages on topics such as: discretionary access control, mandatory access control, content-dependent access control, context-dependent access control, and interdependencies of the Trusted Computer Security Evaluation Criteria (TCSEC) requirements.

The basic taxonomy has since been expanded to include Threat and Security Standards as shown in Figure 3. Appendix A also includes a snapshot of the top level of the INFOSEC Semantic Network. The threat information was obtained from The Taxonomy of Threats and Security Services for Information Systems. [7] This categorization of threat consists of threat consequences and threat actions. The Taxonomy of Threats and Security Service for Information Systems states that threat consequences "define a negative affect that a threat may have on the secure operation of an information system." The four general threat consequences are: Disclosure, Deception, Disruption, and Usurpation. Appendix A displays this portion of the threat model. Threat actions represent scenarios that could cause threat consequences. For example, a threat action of masquerading can cause the consequence of deception. The relationship between security services that counter the generic threat consequence, caused by a threat action, is another important piece of information that is available through the INFOSEC Semantic Network. These relationships are not

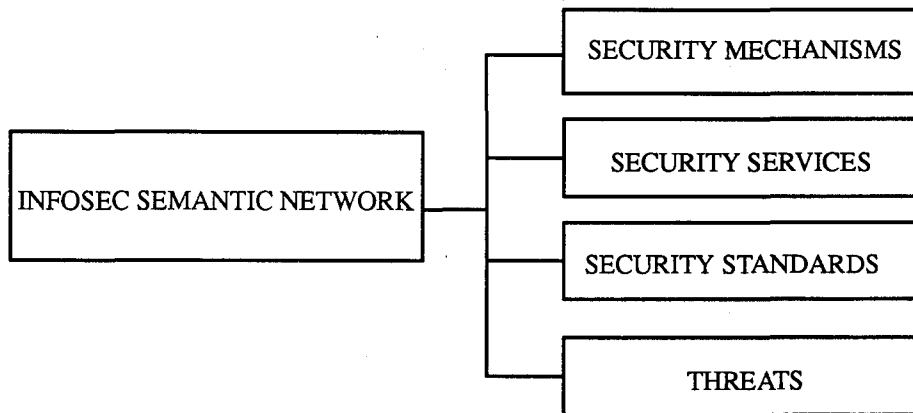


FIGURE 3: TOP-LEVEL VIEW OF THE INFOSEC SEMANTIC NETWORK

one to one and the INFOSEC Semantic Network is able to provide a graphical depiction of the mapping between the security services and threat consequences. This information is likely to be very helpful in completing a system security policy and in determining the security requirements of a system.

The security standards information is based on the National Security Agency (NSA) Open Systems Standards Profile. [10] In the Department of Defense Generic Reference Model there are three platform layers and two interface layers. The application platform is broken down into ten service areas. These service areas are: data interchange, data management, multimedia, network, operating systems, programming, management, real time, security, and user interface. Security standards have been associated with each of these service areas. The INFOSEC Semantic Network provides the ability to retrieve information on security standards and the service area they support. In addition, the connection between the security standards and the associated security service (confidentiality, integrity, availability, and authenticity) is provided. Additional information on each standard includes an explanation of the support it provides to the application platform service area and the security service. Providing this information to ISSEs will enable them to be aware of security standards that NSA has reviewed and determined to be required in order to fulfill our goal of developing open systems.

The INFOSEC Semantic Network uses software from the Central Archive for Reusable Defense Software (CARDS) program. This software, called the Reuse Library Framework (RLF) is Unix-based and government-owned. RLF met an important objective of our program: NOT requiring our users to obtain additional licenses for software or invest monetarily in order to use our product. The RLF capabilities include the ability to search for key words, navigate directly to a node, navigate between parents and children (example: mammals to humans or confidentiality to data confidentiality), display a topography of the network for navigation assistance, graphically display relationships between concepts, and perform actions on nodes including calls of text, program execution, and presentation of audio or graphics. All of these capabilities are displayed in a consistent graphical user interface which maintains a top menu level of six icons.

The development of the INFOSEC Semantic Network began in November of 1992 with RLF version 3.1. The development effort focused on providing information on security services and security

mechanisms and their relationships. After approximately three months of devoted development effort, the proof-of-concept version was presented to five program sites, with approximately 15 individual users, within the NSA beginning in May of 1993. The timeline for these events is depicted in Figure 4. The goal of the proof-of-concept test was to determine if our customers reacted

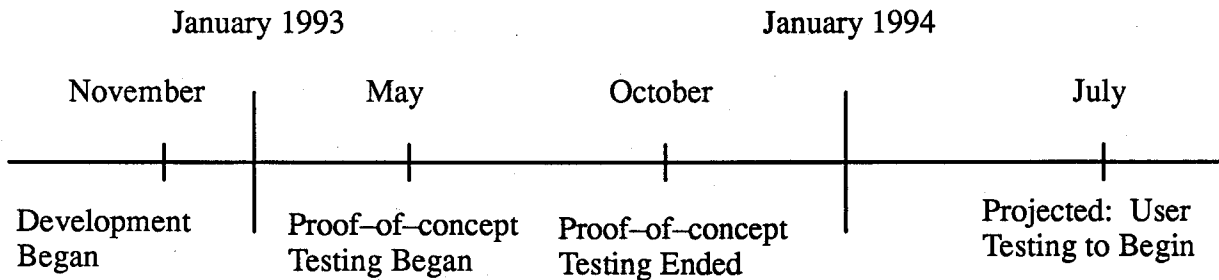


FIGURE 4: DEVELOPMENT TIMELINE FOR INFOSEC SEMANTIC NETWORK

positively to the retrieval of electronic information in this fashion and if the information base we provided was helpful in the design process. The feedback we received was overwhelmingly favorable. For example, responding to our questionnaire, one of the system engineering offices stated that the INFOSEC Semantic Network was, "extremely helpful in clearing up the relationships between services and mechanisms." Another common statement we received from our customers was that the INFOSEC Semantic Network "serves well as a completeness check for requirements hierarchy." Based on customer feedback, we have continued development and have expanded into areas of security standards and threat. We are now preparing for a more extensive user testing phase.

Section 3: Future Testing Plans and Research Directions

In July of 1994, we anticipate providing the INFOSEC Semantic Network to a larger test audience. At that time we will have a product that provides access to INFOSEC domain information on security services, security mechanisms, security standards, and threat. In an approach similar to our proof-of-concept testing we are actively soliciting interested programs to use our tool in the design of their system. To maximize the benefits of Version 2 of the INFOSEC Semantic Network, the test project should be in the early design stages. We will work directly with our customers to provide training on the use of the semantic network and we will provide continual user support throughout the duration of the test project.

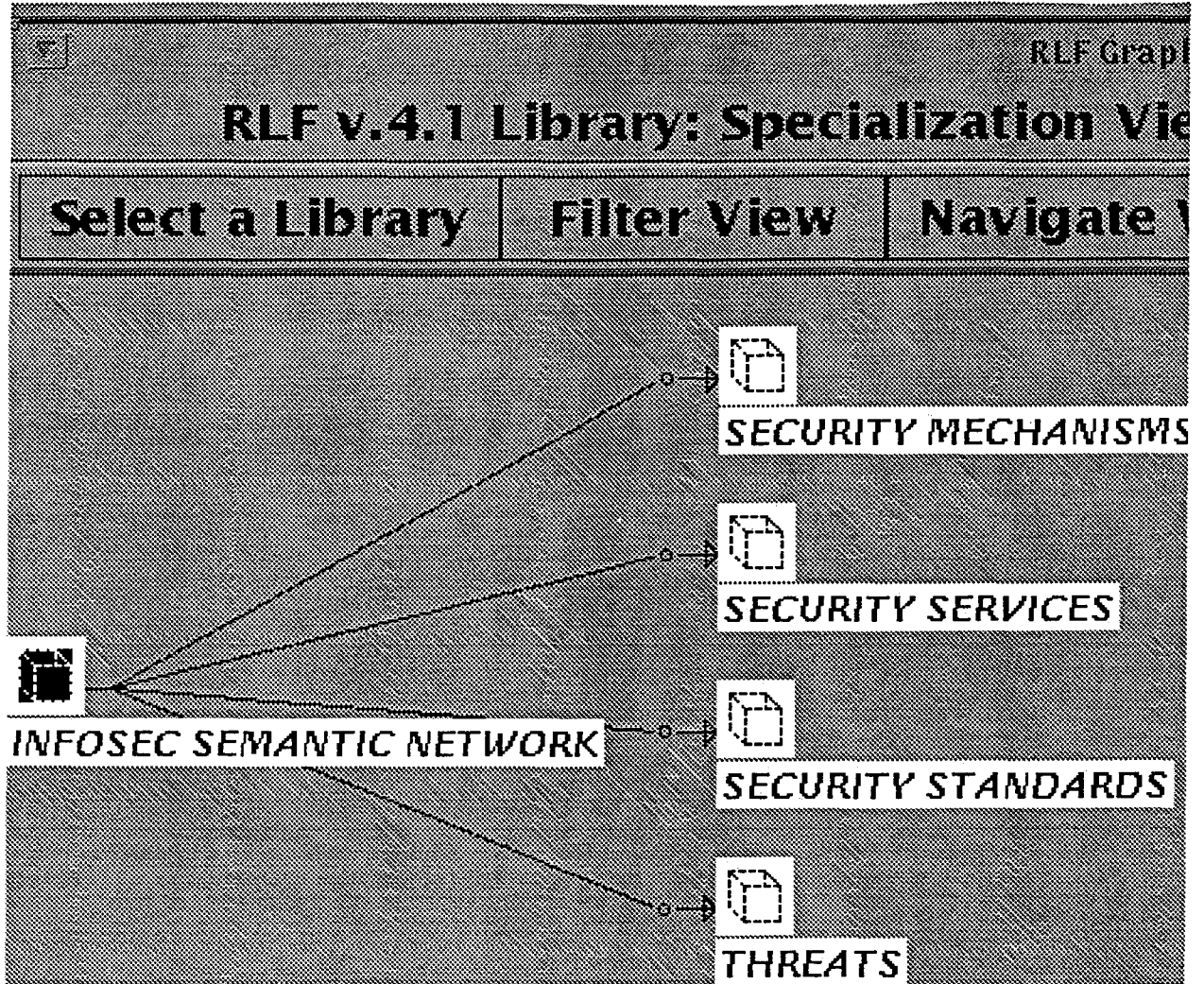
The original test audience documented in their feedback that the INFOSEC Semantic Network was a very useful method of representing INFOSEC knowledge. We expect that the additional capabilities of Version 2 will serve to increase the value of this tool. The possibilities for the expansion of the semantic network are enormous. Our intention is to widen the applicability of the tool to other areas of the secure design process. For example, by capturing detailed design information and providing connections to various guidance documents and available tools we could provide the designer with access to information that would assist in all aspects of the systems engineering process. We will rely on feedback from the next set of test projects to guide research and development in the future.

REFERENCES

- [1] Anderson, J.R. & Bower, G.H. (1973). Human Associative Memory. Holt, New York: Wiley.
- [2] Arango, G., & Prieto-Diaz, R. (1991). Part 1: Introduction and Overview, Domain Analysis Concepts and Research Directions. Domain Analysis and Software Systems Modeling, (p. 9-26). Los Alamitos, California: IEEE Computer Society Press.
- [3] Brachman, R.J. (1979). On the Epistemological status of Semantic Networks. In N.V. Findler (Ed.), Associative Networks: Representation and Use of Knowledge by Computers, (pp. 3-50). New York: Academic Press.
- [4] Brinkley, D., Creps, R., & Badger, L. (1989). A Methodology for the Development of Application-Specific Security Models for Command and Control Systems. Annual Progress Report, 17 March 1989.
- [5] Churcher, P.R. (1989). A Common Notation for Knowledge Representation, Cognitive Models, Learning, and Hypertext. Hypermedia, 1(3), 269-289.
- [6] Creps, R.E. (1989). A Methodology for Defining Application-Specific Security Requirements for C3 Systems. 1989 IEEE Military Communications Conference, p. 900-904.
- [7] Gulachenski, B.D., & Costa, M.J. (18 January 1994). Taxonomy of Threats and Security Services for Information Systems, Mitre Center for Integrated Intelligence Systems Working Paper, WP 93B0000323.
- [8] Information Systems Security Organization, Office of INFOSEC Systems Engineering, Top Down Architecture Decomposition Definitions, 21 October 1991.
- [9] International Standards Organization 7498, Basic Reference Model for Open Systems Interconnection (OSI), part 2: Security Architecture. ISO 7498-2-1988.
- [10] NSA Open Systems Standards Profile, National Security Agency, Version 1.0, 15 May 1993.
- [11] Smith, G.W., & Williams, J.R. (1994). The INFOSEC Body of Knowledge Framework (IBKF): An Introduction. ARCA Technical Report, 22 February 1994.
- [12] Unified INFOSEC Criteria (UIC), First Deliverable, National Security Agency, June 1993.

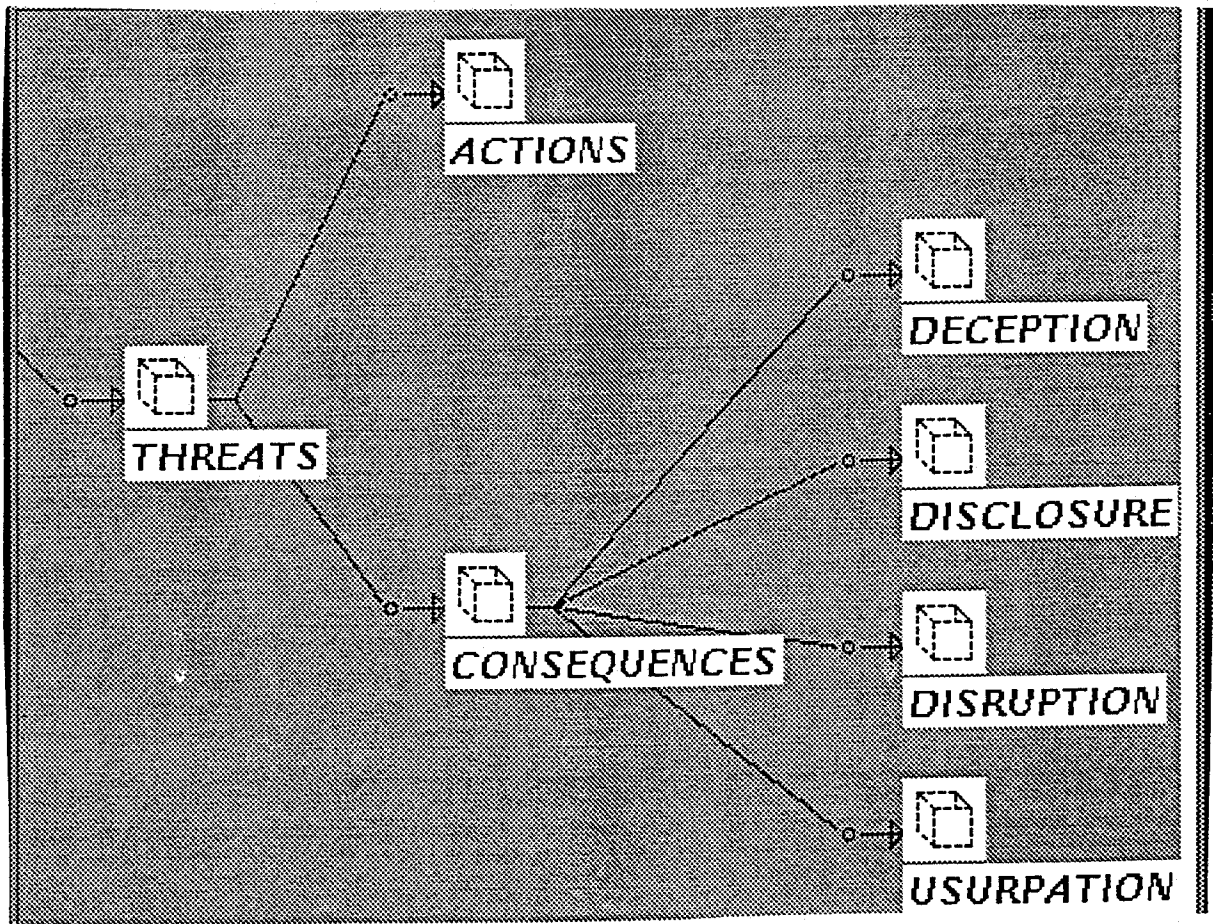
APPENDIX A

TOP LEVEL VIEW OF THE INFOSEC SEMANTIC NETWORK



APPENDIX A

SCREEN CAPTURE OF A PORTION OF THE THREAT MODEL



A New Attack on Random Pronounceable Password Generators

Ravi Ganesan & Chris Davies
Bell Atlantic
Silver Spring, Maryland 20904

ABSTRACT

Given the choice, most users pick poor passwords that are vulnerable to attack. Using random machine generated passwords can ensure that 'good' passwords are chosen, but are user-unfriendly. Machine generated passwords which are 'pronounceable' represent a potential compromise between security considerations and user friendliness. Several such generators have been designed, perhaps the most prominent being the scheme developed by Morrie Gasser [5] in 1977 and which has being recently adopted as a standard by NIST [3].

The security of such generators is typically characterized by the overall size of the password space, which is typically a fairly large number. This is a fairly good security parameter, if the objective of the attacker is to try and compromise a particular account. On the other hand, if an attacker achieves her objective by compromising any account(s) on the system, then the overall size of the password space, in itself, provides an insufficient characterization of the level of security. In fact, as we show in this work, the size of the password space of the pronounceable password generators we examined are fairly huge, yet all suffer from a serious weakness, which allows the attacker to compromise accounts on the system with significantly less effort than the size of the password space would suggest. The attacker cannot choose which accounts to compromise, but in many realistic situations, an attacker's objectives can be met by compromising any account(s).

Conceptually, the password space can be thought of as a large bucket, of size K , from which users pick passwords. It is also true that one can arbitrarily partition this bucket into several smaller buckets, perhaps of different sizes. Consider a small bucket of size b . It might be natural to assume that exactly $\frac{K}{b}$ of the users would pick passwords from this bucket. Unfortunately, in the pronounceable password generators we examine in this work, it so happens that a disproportionately large number of users pick passwords from reasonably small buckets. For instance, in the NIST standard, one such bucket contains only 0.22% of all passwords but it can be expected that about 5% of all users pick passwords from this bucket. The bottomline is that while the NIST standard claims a password space size of "5.7 billion" for 8 character passwords, an attacker who wishes to compromise any 5 user accounts on a multiuser system with a 100 users, need only search through less than 18 million passwords. The impact of the attack depends on the particular implementation and on factors such as 'salting'. Nevertheless, the generators we examined are so acutely vulnerable to our new attack, that we do not recommend that they be used.

KEYWORDS: Dictionary Attacks, Passwords, Random Pronounceable Password Generators, Smallest Bucket Attacks.

Introduction

In this section we chronicle the motivation to use random pronounceable password generators and outline the rest of our paper.

Why Random Pronounceable Passwords?

Poorly chosen user passwords remain a major cause of security intrusions. Attackers typically attack such passwords using dictionary attacks. They obtain, either by eavesdropping on the network, by requesting from a security server (this is possible in the Kerberos[7] system) or from a file on a system, several strings each of which represents known plaintext encrypted with user passwords (e.g. in UNIX a string of zeroes is encrypted with the

user password). The attacker then attempts to decrypt these strings by methodically trying passwords from a dictionary of commonly used passwords, achieving success when a string is successfully decrypted to give the original plaintext. A related approach which uses less time (but more space) is to pre-compute the encryption of all the passwords in the dictionary, so once the strings are obtained, a simple look up is needed to obtain the user password.

There are at least three approaches to solving the problem of poorly chosen user passwords, and each has its field of use. First, smart cards or token authenticators can be used to completely replace the password. Second, proactive password checkers (e.g. [1] or [2]) can be used to filter out poor user choices and force the user to pick a good password. Finally, it is possible to have the system generate passwords for the user. The last approach has two variations:

1. Generate completely random passwords which are by definition guaranteed to be 'good'. This approach has the significant disadvantage of making the password hard to remember, and more liable to be written down (which has a security cost) or forgotten (which has an administrative cost).
2. A compromise approach is to have the machine generate a random, yet pronounceable password for the user, with the assumption that a pronounceable password is more easy to memorize, and consequently more user friendly. Such a system typically works by combining random character generation with the rules for pronunciation to generate strings which are (hopefully) pronounceable. There are at least two important aspects to the design of such a generator. First, are the passwords really pronounceable? Since the so called 'rules' of pronunciation are fairly inexact, this is an extremely subjective issue, and we do not address it further. Of more interest to us is the security of such generators. In the rest of this paper we will illustrate a serious vulnerability we have found in all the pronounceable password generators we have examined including the NIST standard. In a companion paper [4] we describe a scheme we have created which does not appear to have the same weakness.

In this paper we focus on two random pronounceable password schemes, one a scheme used in the version for the Kerberos V source distributed by Sandia National Laboratories, and the second the NIST standard [3] based on Morrie Gasser's[5] system. As we shall show, both these systems have the vulnerability we alluded to earlier. We note that we have observed the same weakness in other pronounceable password generators we have studied.

Finally, it is worth observing that while we have not been able to obtain a security analysis of the Sandia scheme, the NIST system is very carefully analyzed in [5]. In fact, the analysis is among the more complete and comprehensive of security analyses we have seen, and remains a good model of how such schemes should be analyzed. Our conclusion that the Gasser/NIST scheme is insecure (much more so now than in 1977 when it was originally released) is a recognition of the fact that all security systems are eventually broken, rather than a reflection of the care and expertise that went into the original analysis; we reemphasize that we found the original analysis rather impressive in its clarity and comprehensiveness. Also, by publishing the draft standard for a period of review and making source code for the system freely available, NIST encouraged and permitted more detailed analyses, such as the one in this paper. This in effect increases the level of security of the eventual standard, and from the perspective of commercial users of Government standards, we are appreciative of this careful and methodical approach to the standards creation process.

Overview of Paper

In Section 2 we give a generalized description of our new attack. In Section 3 we develop criterion for analyzing password systems, which are useful for ensuring that a system is not vulnerable to the attack we describe in Section 2. In Section 4 we analyze the Sandia scheme against this model and illustrate its vulnerability. In Section 5 we analyze the Gasser/NIST scheme and illustrate why it also is insecure. Finally in Section 6 we conclude by suggesting possible solutions to counter the attack.

The New Attack

Our attack is best described in the form of this simple game. Consider a two dimensional space as represented by a rectangle. Let this password space represent the space of passwords generated by a pronounceable password generator. This is the first figure in Figure 0. Each 'dot' represents a possible password generated. Here is the game:

- Player 1 has to 'color' N dots, these correspond to the N passwords picked by users.
- Player 2 (the attacker) divides the rectangle into B (not necessarily equal) sub-spaces, each containing b_1, b_2, \dots, b_B dots respectively.

If there is a sub-space i that has significantly more than $\frac{N}{B}$ colored dots, then Player 2 wins.

A moment's reflection will show that Player 1 has only one winning strategy, namely, to pick the dots to color at random (i.e. in a uniform distribution).

As we shall see in Sections 4 and 5, the random pronounceable password generators we examine have the unfortunate property that the passwords picked by users tend to be clustered in easily identifiable areas, as a consequence of which, the systems are extremely vulnerable. An attacker does not have to exhaustively search the entire password space, and instead, simply searches a small bucket which contains a disproportionate number of passwords.

How do we protect against such an attack? Firstly, as discussed earlier, an attempt can be made to ensure that users passwords are picked uniformly from all possible choices. One method of achieving this, which may not always be easy to do, is to ensure that all passwords are equally likely to be generated (e.g. see the Gasser variation discussed in the Conclusions). A second method may be to pick a system in which it is difficult for the attacker to discover a 'small bucket'. This is difficult to do, but may well be the only alternative in some cases.

Our depiction of 'dividing the password space' needs clarification. In the Sandia system the division is suggested by the way the system works, namely randomly indexing into one of twenty five buckets from which passwords are picked. In the NIST scheme we divided up the space using the first unit (i.e. passwords beginning with the unit 'a', buckets beginning with the unit 'b', etc.). Both these divisions are arbitrary, and an attacker can choose to divide up the space in any way he chooses (e.g. dividing up the space depending on the unit that appears as the fifth character). As long as a region with a disproportionate number of passwords is discovered, the attacker has succeeded.

Finally, we note that while the disproportionately arises because different passwords have a different probability of being generated, *the key to our attack lies in the aggregate result of being able to locate a sub-space with several such more likely passwords.* Consequently while it may be difficult for an attacker to make a list of 'very likely passwords' for a given system, it may be easier for her to discover a sub-space which contains several such passwords.

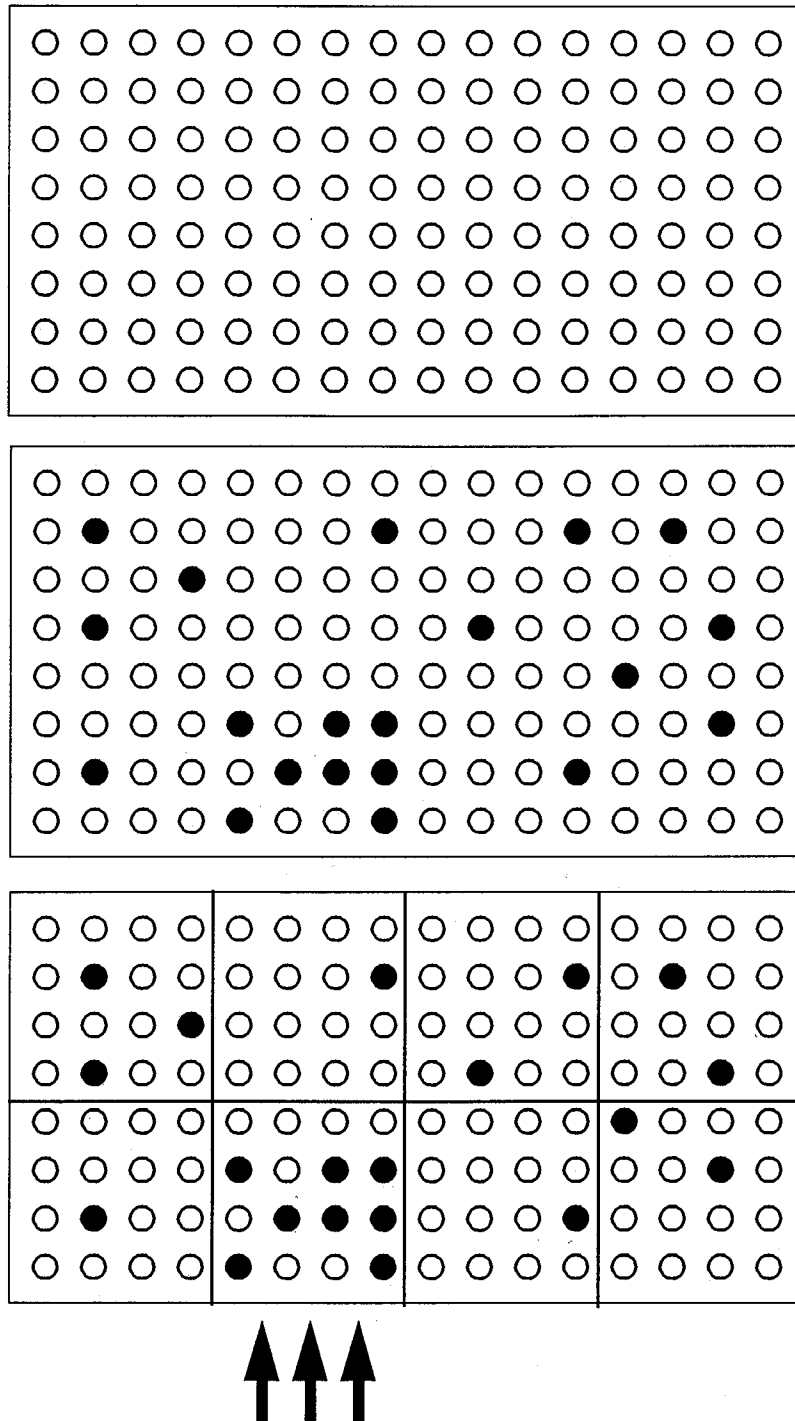


Figure 0: The first rectangle depicts the password space, with each small circle representing a password. The second rectangle shows the selection of passwords (each filled in circle) by users (i.e. the action Player 1 takes). The last rectangle shows one possible way to carve up (i.e. the action Player 2 takes) the password space. In this instance one of the regions (buckets) has a disproportionately large number of passwords, and hence it is profitable for the attacker to exhaustively search this relatively small bucket. This is a simplified example, and in Sections 4 and 5, we will see examples of this attack in which the bucket(s) which the attacker will choose are extremely small relative to the size of the entire password space.

Criterion for Analyzing Password Space Vulnerability

Criterion for the security of password spaces have been fairly well studied, and our intent here is purely to add to, not repeat/replace, well known criterion (for example those that can be found in the Orange book). Our comments apply in general to any password system, though we shall later discuss their applicability specifically to pronounceable password generators. We are mainly interested in the security of passwords in a multiuser security system (e.g. a minicomputer environment, or a Kerberos server serving several users), where the object of the attacker is to:

Attacker Objective 1: break into ANY account(s) on the system

It is our opinion that this is a more realistic formulation of the vulnerability most systems face, than what appears to have been the traditional formulation of the attacker's motive, namely:

Attacker Objective 2: break a PARTICULAR account on the system

While it can be argued that the motives of an attacker will differ for each situation, we believe that any password system must evaluate security in terms of the difficulty of Attacker Objective 1, for two reasons. First, as stated above we speculate that it is the more common attacker motive, and second, a system secure against this attacker objective is automatically secure against Attacker Objective 2, but the converse is not true. Consequently parameterizing the system on the number of users within the 'security domain' being protected is important.

We now define several parameters and explain them:

- K the absolute size of the password space. This is the space the attacker need search in order to break into a particular user's account.
- N the number of users in the 'security domain'. The definition of 'security domain' is situation specific. Some concrete examples would be: a DEC VMS multiuser minicomputer; a network of SUN workstations and servers which use a common /etc/passwd file managed by the NIS name server, a Kerberos realm, serving an entire organization. The number of users within this domain could range from 50 users on a minicomputer to several thousand users being served by a common Kerberos server.
- T the maximum time in seconds we assume the attacker can spend on the attack. T depends on many factors including t , the time interval between which password aging is enforced. When an attacker captures strings encrypted with passwords she has a limited time to complete the attack, before the passwords change. For instance, after time $\frac{t}{2}$, it is highly likely that half the passwords captured by the attacker have changed, and by time t , all the passwords have changed. Depending on the system other factors come into play, and we shall assume that the attacker has a constant time T which she can spend on the attack.
- E the encryptions/second for the particular password scheme, that the most powerful attacker we wish to protect against can afford (or has access to). Since this "access" may well be illegal this is a hard number to calculate. Unless the attacker in question is a large organization, it may be practical to assume that the attacker

has a high end PC or UNIX server at her disposal. We realize that, for instance, specialized hardware for DES encryption has been proposed/built, but we suspect that most attackers do not (as yet) have access to these machines. The figure E can be calculated in various ways, see for instance Karn and Feldmeir's [6] analysis (naturally the actual numbers in that paper are five years old and are obsolete).

- c an implementation specific constant that captures the amount of effort the attacker must expend per user for a specific system. For instance, in UNIX, an attacker searching through a dictionary of size D , would have to, because of the salting [8], actually search through $D \times 4096$ dictionary words. In this case the constant c would be 4096. However, if the attacker is using pre-encrypted dictionaries and has enough space to store all the 'salted' variations, then, at run time, the attacker need expend no further energy (assuming that time to search a list, etc., are small compared to time for encryption) and now c is different. As is clear from this example, this constant needs to be chosen with care, after understanding both the implementation and the possible methods the attacker will use. In the rest of this paper, we will not always explicitly discuss implementation specific details, and it should be understood that individual implementations will have different values of c , and that a password space that we claim is 'small' in general may well be acceptable in implementations where c is large.

Based on these numbers we now define the criterion for protection against attack. The first criterion, which at times tends to be the only one considered by the designers of some systems is:

Criterion 1: $K \gg E \times T \times C$

i.e. they choose a password space that is large enough that it cannot be easily broken by an attacker in a 'reasonable' time. Gasser's analysis adds two closely related, very useful criterion, namely:

Criterion 2: The probability of occurrence of the most probable passwords in the password space should be low

So for instance, although the UNIX password space is very huge, the fact that users pick common natural language words with a very high probability implies that the system by itself becomes vulnerable to dictionary attacks and does not meet this criterion. Gasser discusses this criterion in the context of pronounceable password generators, where as he points out, it is no use if the overall key space K is very huge if a few passwords have a very high probability of being generated, and are generated very frequently by the system. A closely related criterion, which appears to be implicit in Gasser's discussion of the password probability distribution is:

Criterion 3: It is highly desirable that all passwords in the password space be roughly equally probable

This is really a generalization of Criterion 2, and ensures that there does not exist a subset of the password space which can be attacked in lieu of the entire space. Meeting Criterion 3 appears to be difficult (for instance, it is impossible to alter the NIST standard to meet this criterion), and attempting to meet Criterion 4 and 5 might be more realistic.

Criterion 4: In a N user system with a password space of size K , the attacker should have to search through, on average, a password space of $\frac{K}{N}$ in order to break into any one account. Consequently $\frac{K}{N}$ needs to be sufficiently large. This can be expressed¹ as:

$$\frac{K}{N} \gg C \times E \times T.$$

We recommend Criterion 4 be used in place of Criterion 1 since any system meeting Criterion 4, will by definition meet Criterion 1, whereas the converse is not true.

Finally, for pronounceable password generators to avoid the general problem described in Section 2, we describe another criterion.

Criterion 5: It should not be possible to divide the password space into B buckets b_1, b_2, \dots, b_B , with the probability of users picking passwords from the respective buckets being p_1, p_2, \dots, p_B , such that there exists one or more buckets where $p_i \gg \frac{|b_i|}{K}$.

Meeting Criterion 5 is a necessary (but not sufficient) condition for meeting Criterion 4. Further, observe that in cases where Criterion 3 is not met (most realistic cases) meeting Criterion 4 and 5 appear to be fairly good substitutes.

We now turn our attention to the Sandia scheme and the Gasser/NIST systems and illustrate why they do not meet Criterion 5, and consequently Criterion 4, and hence suffer from serious vulnerabilities.

Vulnerabilities in the Sandia System

The system we refer to as the "Sandia System" is a pronounceable password generator distributed by Sandia Labs along with their version of the Kerberos V source code. As of the time of writing this paper we have not been able to obtain further information on the antecedents of the scheme (it appears to have been originally developed by IBM) and we assume that it is used widely within Sandia. We used the files 7c1Cpwd.c and 7c1dpwd.c in Sandia's Kerberos V distribution as our source.

The scheme is fairly simple and works as follows:

- 25 templates have been created to represent typical rules of pronunciation in English. For instance "cvcvcvc" is a template representing words of the form a vowel followed by a consonant followed by a vowel.....
- The templates are formed from sets representing, vowels, consonants, double vowels, ending vowels, etc.
- To generate a password, the system randomly indexes into one of the 25 templates (all 25 templates are equally likely to be picked). We refer to the templates as template-buckets.
- It then picks, at random, a password from that particular template-bucket. This is a 7 character password.

1. Since the attacker need only, on average, search through half any given space to expect to find a password, the more precise figure is $K/2N$.

- In order to inflate the password space, the following is done. Either 1 of 10 digits, or 1 of 26 alphabets, is randomly added to the password, to bring the total password size to 8 characters. If the eighth character is a digit from 0..9, then since there are 10 choices of digits and since the digit can be added in one of eight positions, the password space is expanded by a factor of 80. Similarly, if one of the characters from A..Z is randomly stuffed in, then the effective password space is increased 208 fold.
- Users are presented with several such passwords and asked to pick one.

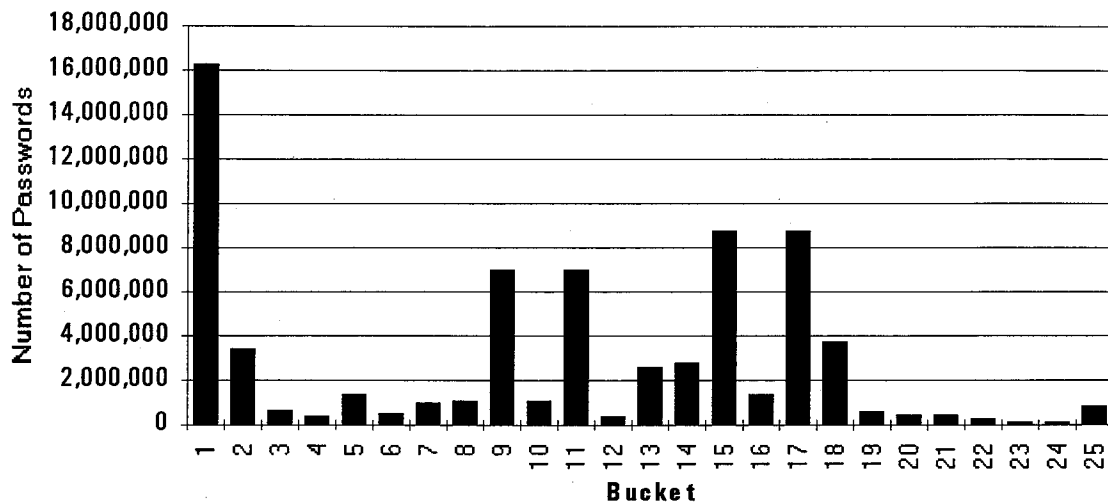
Before proceeding to our main security analysis, we wish to note two points:

1. In our purely subjective opinion, the pronouncability of the passwords generated in the first 7 characters is as good (or as bad, depending on your perspective) as any other generator. However, the addition of the eighth character, in our opinion, makes the result fairly difficult to pronounce. This is especially because the 8th character/digit may well appear in the middle of a pronounceable syllable, thus making the entire word difficult or impossible to pronounce.
2. Presenting users with several choices and letting them pick one, introduces another filter from which passwords get selected. It is conceivable that the passwords picked by users are actually from a much smaller space than would be suggested by the system parameters. We have no way of evaluating if this is indeed the case, and assume that passwords are picked randomly from those presented to the user.

Since the 25 template-buckets are indexed into with uniform probability, it is likely that 1/25 th or 4% of all users in a N user system pick passwords from a particular template-bucket. Given the number of characters in the sets for vowels, consonants, etc., it is trivial to calculate the size of each template bucket. The size of each of the template-buckets (without the addition of the random eight character is shown below in Figure-1.

As can be seen, the distribution is highly non-uniform, with most of the passwords in a few large buckets. This dramatically changes the expected security of the scheme. The total space of 7 character passwords is 71,213,792, and after stuffing the eighth character the total space expands to an impressive 14.5 billion. However, in a 100 user system, 4 users picked passwords from the smallest bucket, which has a mere 135,800 7 character passwords, and after stuffing with the 8th character increases to a password space of 27 million. While an attacker today may balk at searching through 14.5 billion passwords, she can search a space of 27 million without too much effort in order to break into 4 user accounts on a hundred user system (from another perspective: the attacker would on average have to search through less than 3.5 million passwords to expect to break into 1 account on a 100 user system).

Figure 1: Distribution of passwords in template-buckets in Sandia scheme



In our opinion, the net result is that the system may well be more secure with user chosen passwords. For instance, in many systems (though of course not in all) systems administrators have heard enough about dictionary attacks to pick complex passwords. Whereas in using this system they may well pick passwords from small buckets, and in effect the pronounceable password generator has weakened overall security instead of strengthening it.

Finally, wish to reiterate, that the non-uniform distribution of the passwords into buckets, is the main point we wish to note, and the absolute numbers are of less interest since the number of passwords an attacker can actually try will depend on a number of other factors (e.g. salting [8] in UNIX systems).

We now turn our attention to the Gasser/NIST scheme.

Vulnerabilities in the NIST System

The NIST system is far more complex, and harder to analyze, but in effect has exactly the same vulnerability to smallest bucket attacks as the Sandia system. We provide a high level description of the functionality and refer the interested reader to [5] or [3] for more details. The scheme works as follows:

- There are 34 units, the characters A..Z (except Q), CH, GH, PH, RH, SH, TH, WH, QU and CK. Each unit has an associated probability of being picked, which corresponds roughly to the probability of its occurrence in English.

- There are a series of rules for the appearance and positioning of units, and these rules are encoded in two tables - the unit and digram tables. The former describes special rules for where the units may appear, and describes whether they are vowels or consonants, etc. The latter describes the rules according to which two units can be juxtaposed.
- To generate a password the system picks the first unit, from one of the 34 units, based on the probabilities associated with each of the units.
- The system then forms syllables by picking successive units from the list of 34, based on the rules in the unit and digram tables. These syllables are then concatenated together to form the passwords.
- Lastly, we note, that it will often happen that a particular unit that is picked will not be appropriate in a particular place. At this point that unit is rejected, and another unit is picked. If the next unit is also rejected, another unit is picked. This process is repeated 100 times, after which the entire syllable is rejected. As noted in [5] the limit of 100 is rarely reached.

The system is analyzed in [5] and [3], and the following results are obtained:

- The password space is of size 18 million for 6 character passwords, 5.7 billion for 8 character passwords and 1.6 trillion passwords for 10 character passwords.
- The most probable passwords have a low probability of occurrence (see Criterion 2).
- The probability of occurrence of most passwords are (very roughly) equal.

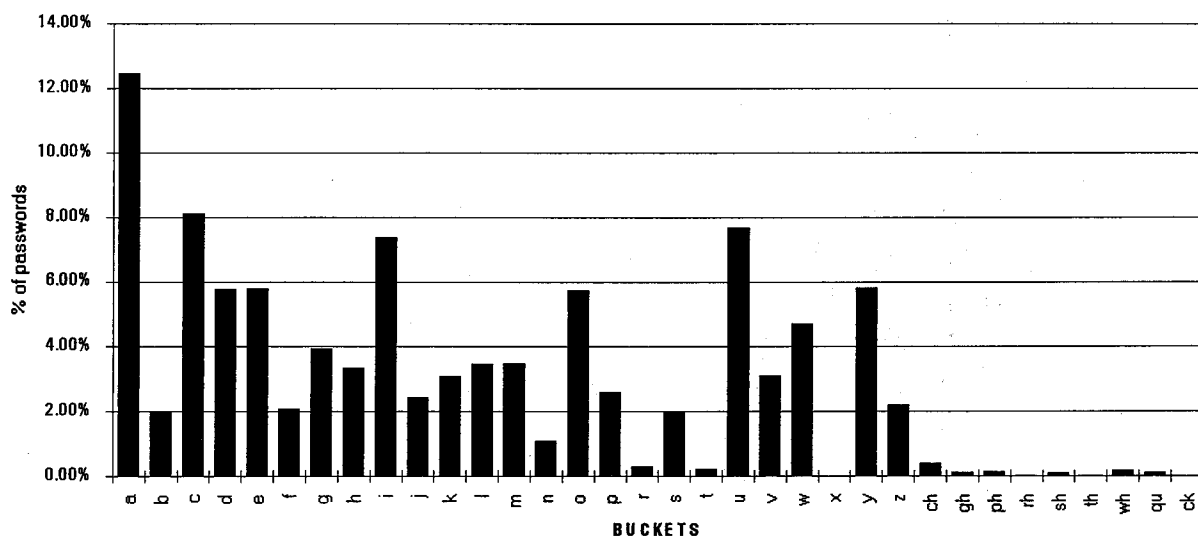
In the Sandia scheme the choice of buckets needed to successfully attack the system was readily obvious. In the Gasser/NIST scheme, the choice is not so obvious, and, as discussed in Section 2, several choices are possible. As it happens, our first attempt at discovering buckets, by differentiating using the initial unit, turned out to result in a successful attack, and consequently we did not look further. It is possible that other choices of buckets might result in a distribution even more favorable to the attacker. So in our attack on the Gasser/NIST scheme each unit represents a bucket of passwords and there are 34 such buckets. However, unlike the Sandia scheme which randomly indexes into the buckets, in the NIST scheme the probability that a user picks a password from a particular bucket is determined by the probabilities associated with the individual unit (since it is very unlikely that a password will not be completed once the initial unit is picked).

Unlike the Sandia scheme it is not easy to directly calculate the distribution of passwords into buckets in the NIST scheme. To calculate this distribution, which is shown below in Figure-2 we resort to the technique Gasser frequently uses in his analysis. In this technique passwords are generated completely at random, and are then passed through the system acting in filter mode, where it determines if the password could have been generated by the generator. As the passwords were generated randomly, sorting the sample into buckets will reflect the actual distribution of passwords into buckets.

Clearly, like in the Sandia scheme, the distribution of passwords is highly non-uniform. However, unlike the Sandia scheme, all the buckets themselves are not equi-probable. Rather, the probability that a bucket is chosen by the system is tied to the probabilities assigned to the individual units. Figure-3 juxtaposes the distribution of the passwords into

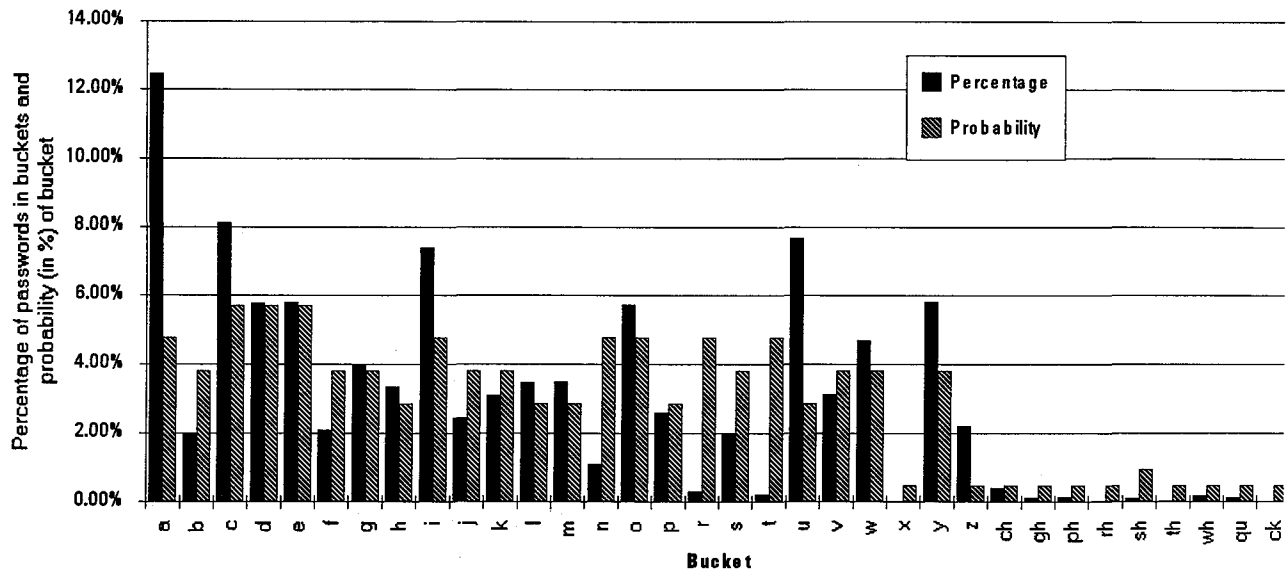
buckets with the probability of a particular bucket (which we can assume is equal to the probability that a user will pick a password from a particular bucket).

Figure-2: Distribution of passwords into buckets in NIST standard.



As can be seen from Figure 2 there are several small buckets (the buckets for R, T, X, GH, SH, TH, QU and CK) but Figure 3 suggests that rather than the smallest bucket itself, it is more beneficial for the attacker to attack the small buckets with a relatively high probability of being chosen (e.g. the buckets for R and T). It is likely that slightly less than 5% of users will pick passwords from the R bucket and another 5% from the T bucket. Yet the size of the R bucket is a mere 0.31% of the overall password space, and that of the T bucket, a mere 0.22%. Consequently, though the NIST standard would suggest a security of a password space of 5.7 billion, for 8 character passwords, an attacker can expect to break into 4 accounts of a 100 account system after searching through a mere 12.5 million passwords, and can expect to break into one account, on average, after searching 1.6 million passwords! Once again, as in the Sandia scheme, we stress that the absolute numbers are less relevant than the non-uniformity in the distribution. The actual numbers will change depending on factors like salting, but the existence of the very small buckets creates a serious weakness, which causes us to suggest that the scheme not be used without some 'fix' to the problem.

Figure 3: Juxtaposition of percentage of passwords in each bucket with probability of the bucket being chosen.



Conclusions: Protecting Against Smallest Bucket Attacks

Here are a number of options to protecting against smallest bucket attacks, listed in no particular order:

1. Use a generator which distributes passwords into buckets uniformly. One such design is described in a companion paper[4].
2. Use the Gasser variant: In this system, passwords are generated at random, and passed through the generator acting as a filter. The distribution of passwords into buckets is still non-uniform, but the probability of a user picking a password from a bucket is exactly equal to the ratio of the size of the bucket to the overall key size, thus meeting Criterion 5. Gasser proposed this system but pointed out that it is somewhat inefficient. Since 1977, when Gasser proposed this method, however, the speed of computers has increased to the point where the inefficiency is a non issue. However, we are extremely reluctant to recommend this as a fix for three reasons. Firstly, we describe the complexity of the Gasser scheme, from a security analysis perspective. We believe that to be 'certifiably' secure, a system must be as simple as possible, so that the design and implementation can be readily analyzed and tested. The Gasser scheme does not meet this test. Secondly, password generators should

be 'portable' across multiple human languages, without requiring significant redesign and recertification. The Gasser scheme is largely English specific. Lastly, in our purely subjective opinion, passwords generated by the Gasser scheme are not sufficiently pronounceable (relative to some other generators we have seen). Since the designs described/referenced in [4] (one our own design, and one a design available from a major manufacturer) are automatically portable to all languages and, in our subjective opinion, generate more pronounceable passwords, we recommend these be considered as serious alternatives to the Gasser variant. We wish to observe however, that having invented one of the suggested alternatives, we may be prone to some amount of bias!

3. Use a password length which makes even the size of the smallest bucket an attacker can find large enough. We have not experimented with 10 character passwords for which the key size is stated to be of size 1.6 trillion. However, assuming the distribution of passwords into buckets is similar to that of 8 character passwords, the attacker would have to, for a 100 user system, on average have to search through 440 million passwords. This may be acceptable. We do not however, recommend this option since subjecting users to 10 character passwords seems a high price to pay for an effective key space size of a mere 440 million.
4. The scheme itself can be fixed to ensure uniformity of bucket sizes by changing the rules in the unit and digram tables. Given the complexity of the Gasser scheme, we would expect that this would be extremely difficult to achieve.
5. It is probably worth investigating the effect of the following experiment: Instead of picking units according to their probability of occurrence in English, pick units with a uniform distribution (i.e. each of the 34 units has a $\frac{1}{34}$ probability of being picked. Since the rules of pronunciation are determined by the digram tables, this will not affect the pronounceability of the resulting passwords. We suspect that this will not solve the problem, but it is an interesting experiment since, in our opinion, the probabilities tied to the units serve no purpose anyway, and a simpler system is easier to analyze.
6. As a quick 'fix' we recommend completely removing all the very small buckets. Observe that this does not solve the problem, and only succeeds in improving the odds against the attacker slightly. The attacker can then switch his attention to one very large bucket, and then focus on the small buckets within that large bucket, e.g. passwords beginning with AR or AT. It is this complication that leads us to recommend that an alternate, easier to analyze, system be used.

Our bottomline recommendation is that NIST adopt a simpler more readily analyzable system which is designed to meet the five Criterion we identified in Section 2.

Acknowledgments

We are extremely grateful to Andy Goldstein of DEC for providing us with the 10 million passwords that we used to perform our analysis of the NIST scheme. We thank Chuck Dinkel of NIST for providing us with useful information on the NIST standard. We are deeply grateful to Raymond Pyle of Bell Atlantic for his insightful review and encouragement. Finally, our gratitude to the anonymous referees for forcing us to carefully rethink our work, and hence make it more coherent.

References

- [1] Bishop, M., "Proactive Password Checking", *4th Workshop on Computer Security Incident Handling*, August 1992.
- [2] Davies, C. and R. Ganesan, "BAsswd: A New Proactive Password Checker", *16th National Computer Security Conference*, September 1993.
- [3] FIPS PUB 181, "Automated Password Generator", Federal Information Processing Standards Publication. 1993.
- [4] Ganesan, R., "BApronounce: A New Random Pronounceable Password Generator", Submitted for publication.
- [5] Gasser, M. "A Random Word Generator for Pronounceable Passwords", *National Technical Information Service (NTIS) AD A 017676*.
- [6] Karn, P.R. and D.C. Feldmeier, "UNIX password security - Ten years later", *Advance in Cryptology - CRYPTO 89. G. Brassard (Ed.) Lecture Notes in Computer Science*, Springer-Verlag. 1990.
- [7] Kohl, J., C. Neuman and J. Steiner, "The Kerberos Authentication Service", *MIT Project Athena (October 8, 1990) Version 5.3*.
- [8] Morris, R. and K. Thompson. "Password Security: A Case History", *Communications of the ACM*, 22(11). November 1979.

DEVELOPMENT HISTORY FOR PROCUREMENT GUIDANCE USING THE TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA (TCSEC)

MAJ (USA) Melvin L. De Vilbiss
National Security Agency
Standards Criteria & Guidelines Division
9800 Savage Road
Fort George G. Meade, MD 20755-6000

In this 10 year effort, through contracts with MITRE, through a National Computer Security Center (NCSC) working group, and now a National Security Agency (NSA) sponsored Procurement Guideline Working Group (PGWG), NSA has produced TCSEC Procurement Guidance, meeting the customer requirements, with Department of Defense (DoD) wide, as well as Federal Government wide, support.

BACKGROUND (see figure 1)

CSC-STD-001-83, Trusted Computer System Evaluation Criteria, was published in 1983. Development of guidance for procurement using this criteria was initiated, under a contract with the MITRE Corporation, that same year. In December of 1985, CSC-STD-001-83 was approved as DoD 5200.28 - STD. By 1985, MITRE had provided two revised drafts of procurement guidance (PG) to the Agency. A third draft was delivered to the Agency in 1986, which was in-turn presented to DoD organizations for review and comment.

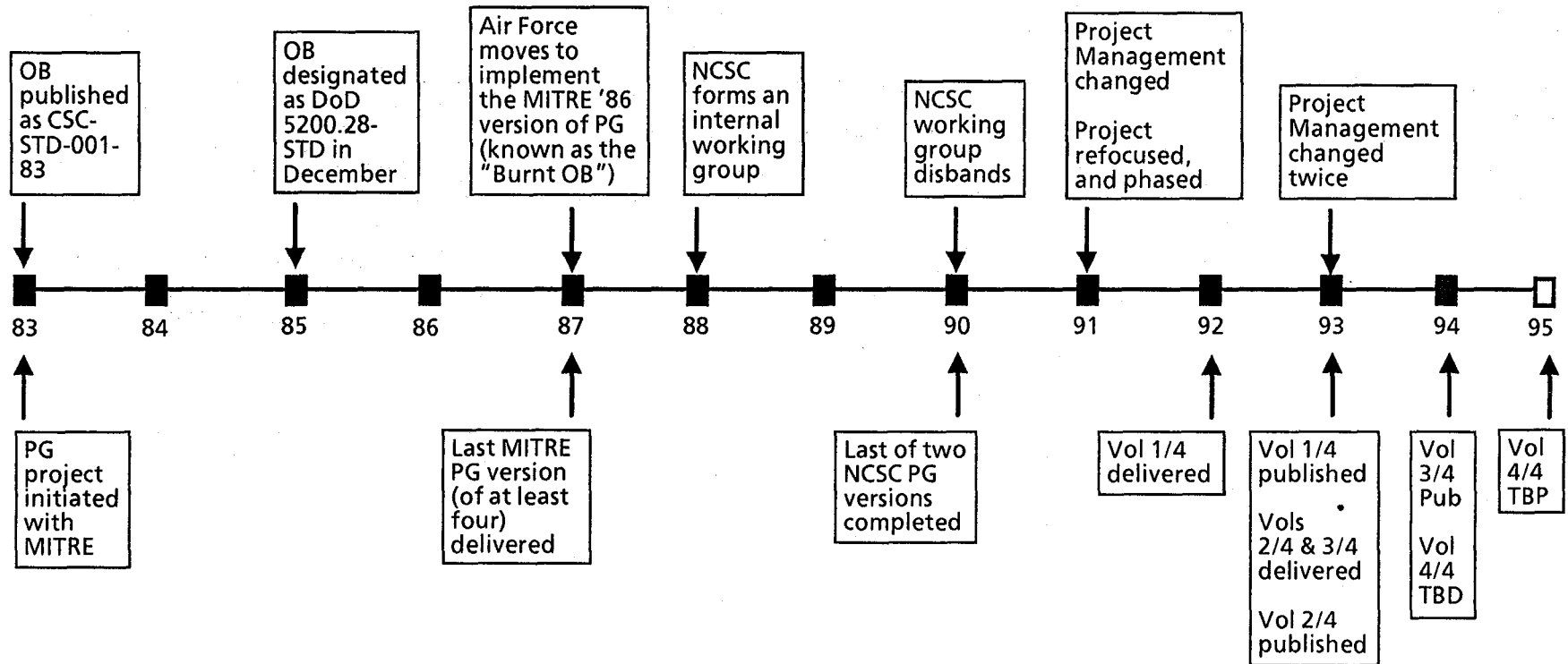
Significant confusion erupted about this third draft that almost destroyed the effort of developing procurement guidance for the TCSEC. To begin the confusion, the Air Force was determined to implement procurement guidance as quickly as possible and attempted to do so by publishing procurement handbooks drawn from the third MITRE procurement drafts. These handbooks eventually became Air Force Security Instructions and Manuals, AFSIs and AFSMs respectively. Their handbooks attempted to deal with a wide range of issues, now known as INFOSEC.

Simultaneous to the Air Force development, issues within the Agency began to flair over the MITRE '86 version. These issues ranged from improper specification of TCSEC requirements, Criteria Creep with requirements specifications, and simply a lack of understanding between COMPUSEC Specialists and Acquisition Specialists. On 24 July 1987, the MITRE '87 version was officially declared unacceptable within the Agency. But the Air Force continued their work with the MITRE '86 version, they later dubbed the "Burnt Orange Book." MITRE delivered a final PG revision to the Agency on 1 Feb 1988.

On 24 June 1988, the NCSC initiated an internal working group to correct the technical problems of the MITRE document and complete the guideline by Sept 1989. On 22 February 1989, the first complete draft was published for internal review. After internal review of this draft, the conclusion was that the NCSC document did not address the target audience properly and had too many holes in it to be effective. A procurement expert was finally brought in to help. A revised, and final, NCSC draft was produced on 20 March 1989. Essentially, work came to a halt with the working group because Data Item Descriptions (DIDs), or the development of such, were either too big or too cumbersome for the working group to handle.

DEVELOPMENT HISTORY FOR THE PROCUREMENT GUIDELINE SERIES

199



DIRECTORY

OB = Orange Book
 PG = Procurement Guideline/Guidance
 TBD = To Be Delivered
 TBP = To Be Published

Grumman Data Systems (GDS) was placed under contract to develop an integrated procurement guideline, using the last NCSC version, on 7 September 1990. The first GDS version was delivered and mailed out for review in December 1990. Part of this version was delivered to the National Institute of Standards (NIST) to assist in the development of a Federal Procurement Guideline titled, *Computer Security Considerations in Federal Procurements: A Guide for Procurement Initiators, Contracting Officers, and Computer Security Officials*, dated March 1992. On 11 Jan 1991, the project was transferred to me.

To bring the project to a successful (i.e. publish procurement guidance) conclusion, I asked that two major decisions be made:

- 1) Define an "integrated system" using the TCSEC requirements;
- 2) If unable to provide this definition, allow me to attack the definition by first providing procurement guidance against the TCSEC, then proceed from there with technical reports and canvassing the users for their experiences, with the eventual goal of providing system level procurement guidance.

The result was:

- 1) We didn't know how to define an integrated system using the TCSEC, and therefore could not use the GDS version of the PG;
- 2) We therefore gained approval to develop guidance in four phases, each phase producing one of each of the four procurement guidelines. First, we were to explain the technical relationship of the TCSEC to the overall acquisition strategy of an Information System. We would then provide Specifications and Statements of Work and develop and catalog a set of DIDs specifically calling out the data requirements of the TCSEC. Finally, we were to provide government evaluation guidance of offerer proposals to assist in the assessment of government requirements satisfaction by the offerers.

By August 1991, the first and third volumes were under development. The final version of Volume 1, borrowed and revised from the Air Force handbooks, was received on 30 Mar 1992, but took until January 1993 to make it through publication editing, management approval and final publication production. On 13 April 1992, Volume 2, a revision of the last GDS version, was approved for development. Volume 4 was placed under contract on 1 June 1992. I was assigned to the Federal Criteria on 9 December 1992, and transferred the project in January 1993.

The final versions of Volumes 2 and 3 (Volume 3 also borrowed heavily from the Air Force handbooks) were delivered in January 1993. Volume 2 took until September 1993 to make it through publication editing, management approval and final publication production. Volume 3 continued to be held up by technical difficulty with the proper wording of DIDs until July 1993, when all DIDs supporting all levels of trust from C2 through A1 were finally approved. It is expected to take until February 1994 to make it through publication editing, management approval and final publication production. In the meantime, the new project officer left government service requiring yet another project management change in June 1993. I returned to the project part time in July 1993. Volume 4 could be concluded in 1994, but it is difficult to make such predictions.

THE PROBLEM

The main reason for past failures of this project was the inability of (Computer) Security specialists to accurately communicate requirements to Acquisition specialists in language that was contractually (legally) binding to the developer.

The eventual success of the project was based on the foundation we built upon lessons learned from past failures. We did this by asking the Armed Services to come together with NSA in a working group to:

- 1) describe past contractual failures at specifying accurate requirements,
- 2) propose solutions,
- 3) develop a solution based on the best proposal.

The proposed solution was complex because:

- 1) there is DoD guidance on the procurement of Automated Information Systems (AISs), DoD-STD-7935A,
- 2) there is guidance on the procurement of software, DoD-STD-2167A,
- 3) there is **no** DoD guidance for the procurement of **security** in AISs.

Therefore, there exists **no basis** for specifying AIS security in contractual language - the problem we had to solve for the customer.

To compound the problem, security in AIS is a subgroup to both AIS development and security (AIS or not) requirements, both of which have extensive DoD requirements specifications. We defined the common ground between the two.

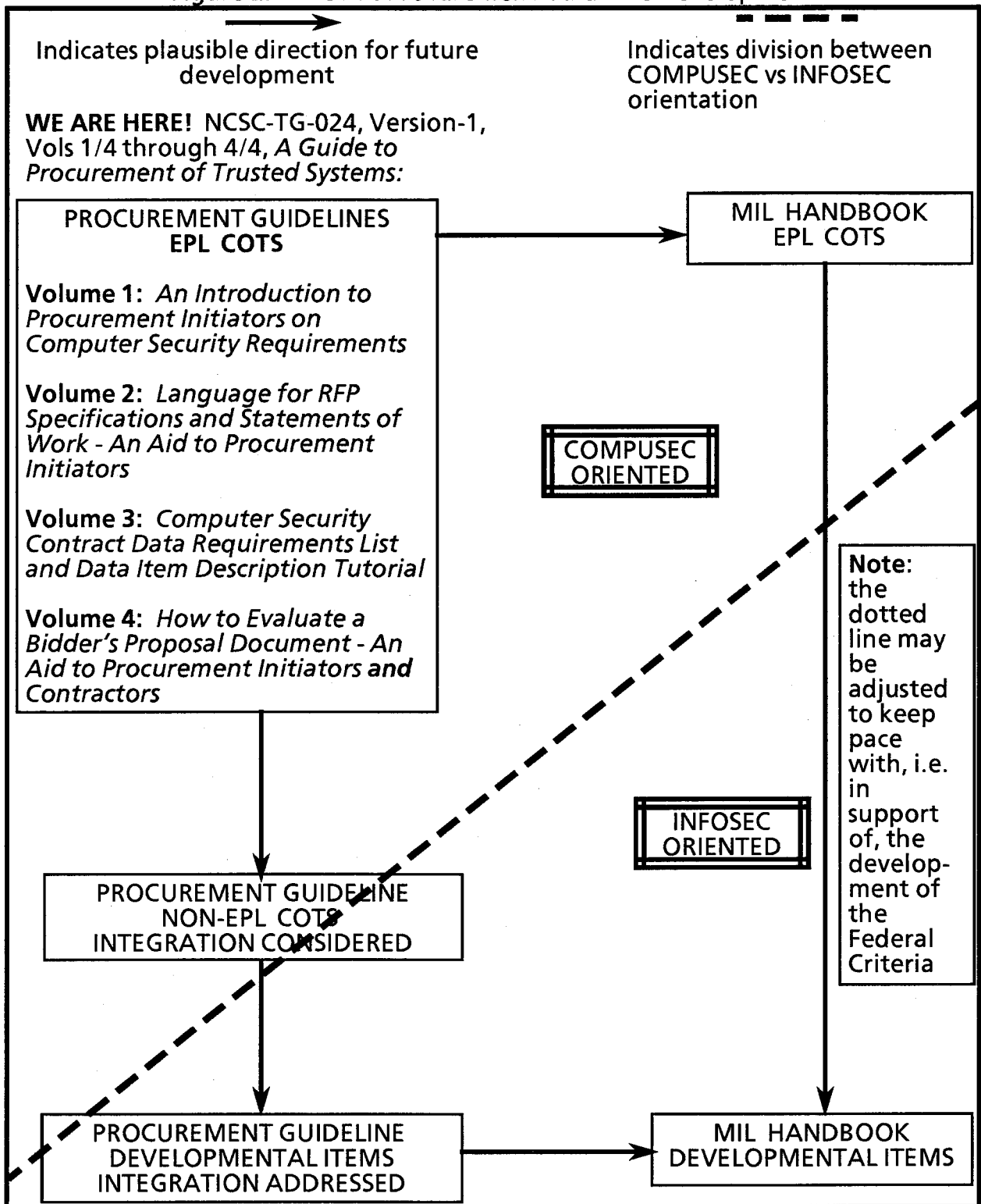
To add one more twist to the problem, security is a requirement which starts at the concept phase of an acquisition and does not end until the final disposition (i.e. disposal) of the product, system, component, etc. In other words, from concept to operational testing, to certification and accreditation, to full scale production, to recertification and reaccreditation, to final disposal, security exists as a requirement which **must** be defined. We defined security in AISs through this entire process.

Finally, Acquisition personnel understand contractual requirements, but rarely understand the totality of what they are requiring in the contract. Hence, the government is replete with Contracting Officer Technical Representatives to assist the contract people with developing the actual language of the contract. In terms of security in AISs, there is **no common ground for understanding** the technical requirements of security in AIS specifications and procurement requirements to make them applicable to both the free and open competition act and to sound, legally binding specifications. We provided this common ground.

THE SOLUTION

The complex solution was to divide the project into four **interdependent** volumes, which must be taken in their totality (see figure 2). Like an interpreter, we devised a common language (the Introduction, Volume 1) for all parties: Acquisition Specialists, AIS security specialists, and contract lawyers. We set the stage by which we could technically, accurately, and legally bind vendors and integrators while satisfying the fair and open competition act. The first document does not provide requirements of any kind. It is a pointer! A pointer to reference manuals, be they acquisition documents, security documents, certification documents, etc. It also points to the succeeding three volumes in this series. It is these three volumes which deal with the technical requirements in terms of procurement format, i.e. Statements of Work, Specification Language, and (peculiar to DoD) Data Item Descriptions.

Figure 2. INFOSEC Procurement Guidance Development



The basis for the introductory document, Volume 1, was the Air Force Handbook. Information Intelligence Sciences (IIS), Inc. worked the handbook from the Air Force perspective up to the DoD perspective.

The purpose of Volume 1 is to advise the procurement initiator that many issues are involved with procurement of AIS security and point him or her to key manuals for additional information. We introduce the Acquisition Specialist to the Computer Specialist, and vice versa, and point them to the following three volumes, as well as to the importance of certification/accreditation planning in the procurement process. Again this document is a pointer. Each chapter has a set of pertinent references to the topic at hand.

Volume 2 is designed to facilitate the contracting process by providing Specification and Statement of Work (SOW) language to procure a trusted system, hopefully satisfied by a product from the NSA Evaluated Product List (EPL). System security requirements are provided in contract language for direct incorporation into a Request for Proposal (RFP). It is technical in nature and duplicates, in a contractual context, the words and intent of the DoD TCSEC. This document is for use by any DoD or non-DoD organization specifying trusted system requirements. For DoD organizations, it specifically calls out Data Item Descriptions (DIDs) unique to the DoD and references Volume 3 for details. The basis for development of Volume 2 spans the entire project from the original MITRE work, through the NCSC working group, to Grumman Data Systems (GDS) for the most significant integration refinements, then eventually to IIS for strictly Computer Security refinements.

Volume 3 provides the DoD officially approved DIDs and explains the packaging procedures for accurately describing the trusted system data requirements. The document is for use by DoD procurement initiators when considering the acquisition of trusted computer products. **Non-DoD organizations will find the Contract Data Requirements List Tutorial informative in that the tutorials provide considerations which should be made while determining the level of trust desired.** Limitations of use of this document are noted, but this guideline is applicable to the data requirements for any acquisition in which security is a factor, whether the procurement is from the EPL or not. The basis for Volume 3 started with the Air Force Handbooks but was greatly massaged through a significant amount of research by GDS.

Volume 4 provides information needed to assist a procurement initiator in developing proposal evaluation criteria, factors, and procedures. This guideline will also assist in the writing of Sections L and M of an RFP for security in AISs. Again, the requirements for this guideline are without regard to an EPL or non-EPL proposal. The basis for Volume 4 lies with the three previous volumes, DoD 5000 series acquisition documents, and DoD Directive 7920.1, *Life-Cycle Management of Automated Information Systems*. CTA, Inc. of Colorado Springs, Colorado is melding the requirements from these sources into Volume 4.

SUMMARY

The gist of this ten plus year effort, is that the Agency started with the TCSEC and attempted to develop procurement guidance for it. MITRE laid the groundwork and suffered the brunt of criticism. The Air force attempted to implement the MITRE version of procurement guidance but learned some hard lessons. The Army and Navy borrowed from the Air Force and continued to learn these hard lessons. NSA plowed on, trying to get a handle on the problem, all the while our customers were

driving through the rain making slow, but never-the-less, steady progress. Then we all got together, put the puzzle in order and produced this guidance, see figure 3. Considering the time, varying levels of support and complexity, I think we owe MITRE and the Air Force a hand for leading the way. Now we need to look at the future and address integrated system procurement guidance.

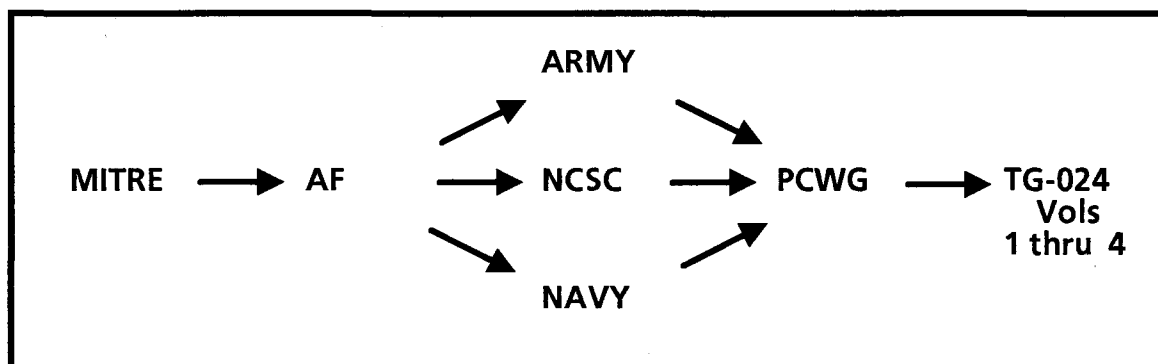


Figure 3. Development of TCSEC Procurement Guidance

CONCLUSION

We've answered the mail for the customers. We have responded to their needs, and done so with their help. They sat down with us in a working group, helped write the documents in formats their people prefer and associate with, and responded to a vendor, DoD, Federal Government, personal user review list over 135 in number, not once, but three times. And all of this since March 1991.

The casual and folksy style (particularly of Volume 1) was the preferred style of the customer, and the majority of reviewers. Given that the last three volumes, by the nature of their information, must be technical, the armed services preferred a casual introduction to the entire process of managing the acquisition of trusted systems.

The response from the field is a resounding "bravo."

Our high expectations of usefulness for these documents are already being met with draft copies of the first three volumes in contractual applications within the services.

REFERENCES

DoD 5200.28-STD, "DoD Trusted System Evaluation Criteria," December 26, 1985.

NCSC-TG-024, Version-1, Vol 1/4, "A Guide to Procurement of Trusted Systems: An Introduction to Procurement Initiators on Computer Security Requirements," December 1992.

NCSC-TG-024, Version-1, Vol 2/4, "A Guide to Procurement of Trusted Systems: Language for RFP Specifications and Statements of Work - An Aid to Procurement Initiators," June 30, 1993.

NCSC-TG-024, Version-1, Vol 3/4, "A Guide to Procurement of Trusted Systems: Computer Security Contract Data Requirements List and Data Item Description Tutorial," February 28, 1994.

NCSC-TG-024, Version-1, Vol 4/4, "A Guide to Procurement of Trusted Systems: How to Evaluate a Bidder's Proposal Document - An Aid to Procurement Initiators and Contractors" (draft).

Exporting Evaluation: an analysis of US and Canadian criteria for trust

Paul A. Olson

As of this year, both the US and Canadian governments have criteria for rating trusted products. While this is a very good development, clearly it would be better to have a single, unified criteria for both countries. This would create one big market rather than two smaller markets, which would benefit both the vendors and their respective governments. Efforts in this direction are underway. In the meantime, can anything be done to allow placing products on both evaluated product lists? This paper will compare and contrast both sets of criteria, analyze the possibility of mapping B1 through A1 requirements to a Canadian equivalent. It will then present some options.

Unbundled Requirements

The first and most obvious difference between the two documents is that the Canadian one is titled "trusted computer *product*" criteria, and the other is titled "trusted computer *system*" criteria. This points to the most fundamental difference between the two: the Canadian requirements are "unbundled", not tied together, so that a product may have different ratings, with different names, for different security features. This approach fits the general term "product". Orange book requirements are very closely coupled under a single name, and are very interdependent; this approach fits the term "system". Any attempt at a mapping must find collections of Canadian criteria requirements that include something corresponding to each orange book criteria class requirement.

Although each document uses different terminology, there is a great deal of similarity in the mechanisms and assurances that they describe. Looking no

deeper than the title of each requirement, one can arrive at the following mapping:

B1:

- T2 Assurance
- Discretionary Confidentiality (CD-2)
- Mandatory Confidentiality (CM-2)
- Object Reuse (CR-1)
- Domain Integrity
- Separation of Duties (IS-1)
- Self Testing (IT-3)
- Audit (WA-2)
- Identification and Authentication (WI-2)

B2:

- T4/5 Assurance
- Covert Channel Analysis (CC-2)
- Discretionary Confidentiality (CD-2)
- Mandatory Confidentiality (CM-2)
- Object Reuse (CR-1)
- Domain Integrity (IB-2)
- Separation of Duties (IS-1)
- Self Testing (IT-3)
- Audit (WA-2)
- Identification and Authentication (WI-2)
- Trusted Path (WT-1)

B3:

- T5 Assurance
- Covert Channel Analysis (CC-2)
- Discretionary Confidentiality (CD-2)
- Mandatory Confidentiality (CM-3)
- Domain Integrity (IB-2)
- Object Reuse (CR-1)

- Separation of Duties (IS-1)
- Self Testing (IT-3)
- Audit (WA-3)
- Identification and Authentication (WI-2)
- Trusted Path (WT-2)
- Recovery (AY-1)

A1:

- T7 Assurance
- Covert Channel Analysis (CC-2)
- Discretionary Confidentiality (CD-2)
- Mandatory Confidentiality (CM-3)
- Domain Integrity (IB-2)
- Object Reuse (CR-1)
- Separation of Duties (IS-1)
- Self Testing (IT-3)
- Audit (WA-3)
- Identification and Authentication (WI-2)
- Trusted Path (WT-2)
- Recovery (AY-1)

Security Features

If a computer customer is trying to compare US and Canadian products, the above list is about as close as one can come to defining what, say, B2 means in Canada, and what approximates a B2 in the US.. However, this is based solely on a comparison of topics. In the actual text of the requirements, there are several crucial differences that must be considered.

The two criteria sets are much alike in the area of assurance and assurance mechanisms (e.g., hardware self-testing). They are very divergent in areas of policy and security mechanisms. CD-2 and CM-2, for example, do not exactly correspond to orange book DAC and MAC. CD-2 states

The TCB shall enforce an approved discretionary confidentiality policy to protect against information disclosure. The approved policy shall define the set of the product's objects to which it applies. [2]

CM-2 states the same thing regarding a mandatory policy. The orange book criteria also state that a policy will be used, but they go on to stipulate what constitutes an approved policy. The MAC requirement in the orange book states:

The TCB shall enforce a mandatory access control policy over allsubjects, objects, and I/O devices...that are .. accessible by [untrusted] subjects. These subjects and objects shall be assigned sensitivity labels that are a combination of hierarchical classification levels and non-hierarchical categories, and the labels shall be used as the basis for mandatory access control decisions.[1]

The requirement goes on to specify the read and write policy. There are also, of course, five to eight additional requirements detailing what labels will look like, how they will be controlled, and what they will be attached to. The orange book DAC requirement likewise specifies what a correct discretionary policy looks like:

[The TCB] shall allow users to specify and control sharing of .. objects and shall provide controls to limit propagation of access rights....[1]

and at B3:

[The] access controls shall be capable of specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective modes of access to that object. [1]

Plainly the two sets of criteria are very different, yet they are also related. Because the US criteria are more specific, and the Canadian criteria more general, one may think of the US MAC and DAC requirements as subsets of CM-2 and CD-2. That is, B1-thru-A1 MAC will meet the CM- requirement, but a CM- rated product will not necessarily meet B1 MAC.

Thus, a customer that for example needs a B1-ish product can avoid those Canadian products that lack a CM- rating, but for those that have CM- rating, he (or someone) must verify that the policy supported by the CM- product is *at least as restrictive* as TCSEC MAC. That is, one that will not permit any

access that would be disallowed by TCSEC mandatory controls involving labels and dominance. The same is true for products with discretionary controls. For example, a policy of total isolation of domains (appropriate for some applications) is more restrictive than TCSEC MAC, and could be interpreted to meet the requirement.

Identification & Authentication vs. WI-

These requirements map very well; both require that users be authenticated before logon, that the authentication mechanism will be protected, and that certain attributes will be attached to them to assist in the enforcement of security policy(s). However, because the TCSEC requirement is tied to other requirements, it is more specific about what those attributes will be. In this case, a label range and a unique, auditable identity are required by TCSEC I&A, where the WI- requirements stipulate simply unspecified "attributes".

Again, a TCSEC-rated product will meet the WI- requirements. A WI- rated product might also meet TCSEC I&A, but further analysis will be needed to verify this.

Audit vs WA-

TCSEC audit and CTCPEC WA- both require a protected, accurate audit trail that enables tracking of users' actions and certain attributes about them (date, time, etc.). The difference, again, is in the details: TCSEC audit gives a list of required auditable events before saying "and other security-relevant events [1]". The WA- requirements allow the submitted audit policy to define the security-relevant events, and do not require that process/object labels be included in audit records. Again, this is because audit is not bundled together with any labelling requirements. Interestingly, the WA- requirements explicitly mandate the presence of audit reduction tools, which are only implicitly required (i.e., by interpretation) in the TCSEC.

TCSEC audit will meet WA- requirements. WA- rated products will meet TCSEC audit if the list of events includes the necessary items, and the audit records include all relevant labels.

Object Reuse vs. CR-1

These two requirements are nearly identical, except that the TCSEC again adds a detail, that not even "encrypted representations of information [1]" are to be available in reallocated objects. Without those words, presumably CR-1 could be interpreted to regard encrypted data as "unavailable." Alternatively, CR-1 could forbid it on the grounds that it is not an "acceptable object reuse policy [2]".

Trusted Path vs. WT-

Both of these requirements prescribe a reliable, unforgeable communication path both prior to logon and whenever a user-to-TCB channel is needed. As this is a relatively simple mechanism, the two requirements not surprisingly map very well, and may be regarded as equivalent. According to the table

above, therefore, B2 Trusted Path equals WT-1, and B3/A1 Trusted Path equals WT-2.

Assurance

The assurances of each criteria set also map rather well with one another, especially as the Canadian criteria do not unbundle the assurance requirements to the same degree as the security features. The “intrinsic” assurances (the ones that improve the process of design and development of the product) are rolled into one requirement set, called T-1 through T-7. Other forms of assurance such as domain separation and product self-testing are broken out into separate requirements, presumably because they are more “active”, i.e., they provide assurance of correct operation while the system is running. Still other forms of assurance have no correpondance at all in the TCSEC, such as resource containment, fault tolerance and robustness, and are not addressed in this mapping exercise.

However, the CTCPEC assurances are bundled together in a different manner than TCSEC assurances. For example, the IB- or domain isolation requirement maps to a portion of the TCSEC system architecture requirement. Both require the TCB to have “a domain for its own execution that protects it from external interference and tampering”. However, the same system architecture requirement also stipulates how the TCB will be designed and structured. This is done elsewhere in the CTCPEC, in the “Architecture” section of the T-1 through T-7 requirements. Again, this probably reflects the view that domain isolation is a separate, pro-active protection mechanism, and good architecture is more an intrinsic characteristic of a secure product.

In addition, the different sets of assurances are like the different sets of features in that the TCSEC is more particular and the CTCPEC more general. This again reflects the fact that the TCSEC is addressed to a particular type of product, general-purpose operating systems, where the CTCPEC wishes to address a wider variety of product types. Some specific differences, and similarities, are described below:

System Architecture

B1 system architecture maps fairly well to IB-2 and T-2 (Architecture), except that B1 stipulates “process isolation through the provision of distinct address spaces under [TCB] control [1]”. The first paragraph of IB-2 requires that the domain policy “identify the TCB domain *and other domains* [2]”. If a vendor’s domain policy identifies “other” domains as process address spaces, then the two requirements map cleanly. Also, the TCSEC requirement says that the TCB “shall isolate the resources to be protected so that they are subject to the access control and auditing requirements”, where T-2 Architecture says simply “The TCB shall enforce the product’s security policy”. Allowing for the unbundling principle, the two requirements are nearly the same, again except for the bit about isolation. When mapping a system’s ratings, therefore, some additional analysis should be done to verify that

domain and resource isolation are used to meet the CTCPEC requirements, thereby matching the TCSEC requirements. Note that in the type of system the TCSEC was designed to rate, it is hard to envision how else to properly control resources other than to isolate them. Consequently it will probably not be major issue.

At B2, the system architecture requirement adds hardware support for the protection mechanisms, least privilege, and a TCB made of “well-defined, largely independent modules [1]”. In the CTCPEC, the T-3 architecture requirement says that the TCB be made of “well-defined, largely independent components [2]”, T-4 requires the use of protection mechanisms in “the underlying abstract machine”, and T-5 requires the principle of least privilege. The T-5 architecture requirements, therefore, map best to B2 system architecture. The only issue is whether an “abstract machine [2]” maps to “hardware[1]”. Clearly it can, but again a given rated product must be analyzed to verify this. Interestingly, T-5 architecture also adds minimal complexity, a conceptually simple protection mechanism, and “modularity, abstraction, and data hiding”, roughly analogous to B3 system architecture. The only quibble is that B3 says “layering” rather than “modularity”. Layering is a concept most relevant to operating system design, and was probably thought too specific for the CTCPEC. A CTCPEC-rated product would have to include layering during initial design, in order to fully map to the B3 requirement.

System Integrity

The B1-A1 system integrity requirement maps virtually one-for-one to the CTCPEC IT-3 requirement. The IT-3 requirement even includes explicitly what US evaluators require by interpretation.

Covert Channel Analysis

The CC-2 requirement maps most closely to B2 covert channel analysis. It actually requires more than B2, because the CTCPEC does not make the distinction between storage and timing channels that the TCSEC does. However, it also does not impose any explicit bandwidth restrictions other than a vendor’s estimate of the maximum allowable by the product’s intended environment. A B2 system would need to be analyzed for timing channels to fully map to CC-2, and a CC-2 product would need to verify that the maximum bandwidths of all channels are within TCSEC limits.

At B3, the timing/storage distinction goes away. Bandwidths will still have to be compared. At A1, formal methods must be used in the analysis. No CC-requirement calls for this. A vendor would have to undertake covert channel analysis with this in mind in order to map the two requirements.

CC-3 calls for elimination of covert channels, which is implicitly considered impossible in the TCSEC. There is no mapping here.

Separation of Duties

The IS-2 requirement maps virtually one-for-one with the B2 and B3 trusted facility management requirements. B3 requires that assuming an administrative role be an auditable event where IS-2 does not, another side-effect of unbundling. If the IS-2 product contains a WA- rating, it can be analyzed for this extra requirement.

Trusted Recovery

B3 trusted recovery maps best to the AY-1 requirement. AY-1 actually contains more specific requirements than B3. For example, it requires the vendor to define failures and discontinuities from which recovery is possible, and to establish failure thresholds beyond which the product must be re-installed. A B3 system must be analyzed for these qualities before a mapping can be established.

Other Assurances

All the other assurances in the TCSEC, trusted distribution, specification & verification, configuration management, security testing, and documentation, have counterparts in the T-2 through T-6 requirements that map with little difficulty. The terms and phrasing are very similar in both documents. No additional analysis is needed in these cases to establish a mapping.

Evaluation Options

Given the technical issues outlined in this paper, several options present themselves in the case of vendors who would like to be on both North American evaluated products lists. Evaluation agencies in both countries can elect to do one of the following with the other's ratings:

(A) Nothing. Re-evaluate the system according to one's own criteria from scratch. This describes the present situation. It has the advantage of consistency with one's own evaluation program, but is highly undesirable for many

reasons. It unnecessarily constricts the market for trusted products, it wastes time and money, and it virtually insults one another's technical expertise.

(B) Translate. Refine a mapping process and translate ratings from one criteria to the other. This would require altering or adding to the evaluation programs of each country. It has the advantage of reducing customer misunderstandings, and saves time and money by re-using each other's analysis. It would still require some effort to perform the extra analysis of the sort outlined in this paper, however. It also might effectively make vendors who want to sell in both countries build their products to two different standards simultaneously. This would make them more costly, particularly at higher levels of trust.

(C) Publish. Simply post one another's rated products as a separate part of one's own evaluated products list, and optionally publish some document to help customers make sense of the other's ratings. For the evaluator agencies this is the least costly option of all, requiring no additional analysis or mapping. For vendors it increases customer base without having to build to two standards. For the customers it increases the number of types of rated products, although it also requires them to understand two different rating schemes.

Conclusion

Although the two North American criteria sets differ in underlying philosophy, rating scheme, and other attributes, the desired result is the same: to make computer security products widely available. To that end it is clearly desirable to create as wide a market as possible. Prior to the publication of a Common Criteria, an in-depth comparison of the two criteria sets might open the way for mutual recognition of one another's evaluation work. Further analysis might be pursued in comparing TNI and TDI ratings with the CTCPEC.

Bibliography

- [1] Trusted Computer System Evaluation Criteria, 1985, DOD 5200.28 STD
- [2] Canadian Trusted Computer Product Evaluation Criteria, Version 3.0e, 1993, The Government of Canada

WHAT COLOR IS YOUR ASSURANCE?

David R. Wichers, Joel E. Sachs
Arca Systems, Inc.
10320 Little Patuxent Parkway, Suite 1005
Columbia, MD 21044
wichers@arca.md.com
sachs@arca.md.com

Douglas J. Landoll
Arca Systems, Inc.
8229 Boone Blvd., Suite 610
Vienna, VA 22182
landoll@arca.va.com

Abstract¹

This paper compares and contrasts a number of prominent development and evaluation methods to illustrate that the type of assurance they provide varies. It then presents a taxonomy of assurance which clarifies the different assurance aspects of these development and evaluation methods. This taxonomy is intended to help the community recognize, understand, and focus on particular assurance issues by facilitating the definition, discussion, and comparison of the type of assurance provided by existing or proposed development or evaluation methods.

Keywords

Assurance, Development Methods, Evaluation Methods, CMM, TSM, TCSEC, ITSEC, Federal Criteria, CTCPEC, RAMP, ISO 9000, CCEP, Profiling, Certification, Accreditation, Security Engineering Capability Maturity Model (SE CMM)

Introduction²

A common misperception exists that assurance has a single meaning, purpose, or goal. People tend to lose sight of the fact that there are many different types of assurance and the need for assurance varies. This paper discusses this misperception and presents a taxonomy which will clarify the different aspects of assurance and help the community recognize, understand, and focus on particular assurance issues.

To quote Webster's, assurance is "something said or done to inspire confidence" [1]. Inspiring confidence is the fundamental part of assurance that is intuitively understood. People also understand that confidence is inspired by different methods. What they tend not to understand is that these methods inspire different types of confidence and may even be targeted at different types of products or systems. Contributing to this confusion is the fact that many of the methods for providing assurance are similarly organized into levels of improvement, which makes people think they are equivalent. Some methods even have misleading names, which is also confusing. The focus of this paper is to eliminate this confusion and present a taxonomy which facilitates defining,

¹ The title of this paper was inspired by the book "What Color is Your Parachute?" by Richard Bolles. [2]

² This research was supported by the National Security Agency under Contract Number MDA904-93-C-C029. This work was accomplished as part of a joint effort by NSA's Systems Security Engineering Division, Arca Systems, and CSC Professional Services Group to develop a Security Engineering Capability Maturity Model.

discussing, and more accurately comparing the type of assurance provided by the various development or evaluation methods.

Background of Research

The authors are part of a team that is developing a Security Engineering Capability Maturity Model (SE CMM) [3]. This model, which is based on the Software Engineering Institute's Capability Maturity Model (CMM) [4], is intended to help security engineering organizations define and improve their security engineering process, much in the same way as the CMM promotes process improvement for software engineering organizations. As part of developing the SE CMM, we needed to understand how different development and evaluation methods provide assurance since assurance is a key aspect of security engineering and is not present in the CMM. This research helped us gain this understanding, and has influenced the practices in the SE CMM for improving an organization's ability to provide assurance throughout their security engineering process.

Research Approach

To help identify and illustrate the different aspects of assurance, we examined the prominent development and evaluation methods which are listed and briefly described in Table 1. To facilitate this examination, we characterized each of these methods in order to be able to compare and contrast them. The key to the characteristics we used is presented in Table 2 and the characterization of each method is presented in Table 3.

Table 1: Development and Evaluation Methods Examined

Short Name	Name of Method	Description
CMM	Capability Maturity Model for Software	SEI's Software Development Process Improvement Methodology
TSM	Trusted Software Methodology	SDIO's Methodology for Developing Mission Critical Software
TCSEC / TPEP	Trusted Computer System Evaluation Criteria, Trusted Product Evaluation Program	DoD Trusted Product Criteria and Evaluation Methodology
ITSEC / ITSEM	Information Technology Security Evaluation Criteria, Information Technology Security Evaluation Method	European Trusted Product and System Criteria and Evaluation Methodology
Federal Criteria	Federal Criteria for Information Technology Security	Proposed U. S. Trusted Product Evaluation Criteria
CTCPEC	Canadian Trusted Computer Product Evaluation Criteria	Canadian Trusted Product Evaluation Criteria
RAMP	Ratings Maintenance Phase Program	TPEP Product Rating Maintenance Program
ISO 9000 Series	Quality Management and Quality Standards	ISO Quality Assurance Standards
CCEP	Commercial COMSEC Endorsement Program	NSA COMSEC Evaluation Methodology
Profiling	Center for Profiling	NSA Trusted Product and System Assessment Methodology
Certification	Certification of Government systems	DoD System Procurement Requirement Verification Methodology
Accreditation	Accreditation of Government systems	DoD Decision to Operate Methodology

Table 2: Characteristics of Development and Evaluation Methods

Characteristic	Description of Characteristic
Introduction Date	When the method was first introduced.
Purpose of Method	What the method was intended to explicitly achieve.
Intended Target of Method	What it was intended to be applied to (e.g., all products, trusted products, systems, software).
Premise of Method	What it was hoped the purpose of the method would ultimately achieve.
Strategy of Method	How the method attempts to achieve its purpose.
Criteria Organization	How the criteria portion of the method is organized.
Requirements (broken into the following areas:)	Each table entry indicates that requirements are levied in that area by the criteria. The text of the entry describes how compliance with those requirements are actually evaluated.
-Target Arch	The architecture of what is being built
-Target Design	The design of what is being built
-Target Implem	The implementation of what is being built
-Assur Evid	The assurance evidence produced throughout development
-Eng Proc	The engineering process used throughout development
-Eng Envir	The physical and computational environment in which development is done
-Target Envir	The environment in which the product or system is to be, or is being, fielded
-Risk of Use	The risks of using what is being built
Purpose of Evaluation	Why the evaluation is done.
When Evaluated	When in the lifecycle the evaluation is performed.
How Evaluation is Performed	How and where the evaluation is done. Where is relative to the development location.
Who Performs Evaluation	Who performs the evaluation.
Result of Evaluation	The quantifiable result which is made available after the evaluation is complete.

Description of Development and Evaluation Methods

The following four part table describes each development and evaluation method listed in Table 1. The key to the characteristics is presented in Table 2. It is intended to represent the essence of each method rather than every detail. A specific set of terminology is used as well, which is not always the same terminology used by every method described. This was done to make the terminology in the table as consistent as possible, and to facilitate the ability to analyze, compare, and contrast the methods.

Table 3: Description of Development and Evaluation Methods (Part 1)

	CMM	TSM	TCSEC / TPEP
Introduction Date	1986	1991	1983
Objective of Method	<ul style="list-style-type: none"> • Provide Guidance for: <ul style="list-style-type: none"> - Defining - Improving a software development organization's software development process • Provide metric for identifying qualified software contractors 	<ul style="list-style-type: none"> • Provide: <ul style="list-style-type: none"> - Guidance for establishing - Metric for evaluating - Basis for specifying safeguards to be integrated into the software development process 	<ul style="list-style-type: none"> • Provide: <ul style="list-style-type: none"> - Guidance for developing - Metric for evaluating - Basis for specifying COTS trusted products
Intended Target of Method	<ul style="list-style-type: none"> • All Software 	<ul style="list-style-type: none"> • Mission Critical Software <ul style="list-style-type: none"> - for Strategic Defense 	<ul style="list-style-type: none"> • COTS Trusted Products <ul style="list-style-type: none"> - for DoD Environments
Premise of Method	<ul style="list-style-type: none"> • Improving process will: <ul style="list-style-type: none"> - improve quality - reduce cost - shorten schedule of software projects 	<ul style="list-style-type: none"> • Strict controls will reduce the potential for both: <ul style="list-style-type: none"> - malicious and - inadvertent subversion of the software 	<ul style="list-style-type: none"> • A common criteria will: <ul style="list-style-type: none"> - make trusted products more widely available - provide a basis for: measuring assurance specifying security requirements
Strategy of Method	<ul style="list-style-type: none"> • Criteria defines: <ul style="list-style-type: none"> - organizational structure - management structure - fundamental engineering activities necessary to achieve higher levels of maturity. • Criteria used to assess status of process maturity and suggest improvements 	<ul style="list-style-type: none"> • Criteria defines specific levels of control on: <ul style="list-style-type: none"> - engineering environment - activities in engineering process • Independent evaluators monitor and document degree of compliance with targeted level 	<ul style="list-style-type: none"> • Criteria defines specific levels of: <ul style="list-style-type: none"> - security features - assurance of COTS trusted product • Independent evaluation ensures compliance with targeted level
Organization of Criteria	<ul style="list-style-type: none"> • 5 Hierarchical levels <ul style="list-style-type: none"> - Levels 1 through 5 of - increasing process maturity 	<ul style="list-style-type: none"> • 6 Hierarchical levels <ul style="list-style-type: none"> - T0 through T5 of - increasing environmental and procedural controls 	<ul style="list-style-type: none"> • 6 Hierarchical levels <ul style="list-style-type: none"> - D through A1 of - increased features bound to - increased assurance
Requirements	How Evaluated	How Evaluated	How Evaluated
-Target Arch			Analyze
-Target Design			Analyze
-Target Implem			Test
-Assur Evid			Analyze
-Eng Proc	Confirm Process Exists	Confirm Process is Followed	
-Eng Envir		Confirm Controls are Used	
-Target Envir			Somewhat addressed in Yellow Book
-Risk of Use			Allowed risks described by Yellow Book
Purpose of Evaluation	<ul style="list-style-type: none"> • Identify Current Maturity Level • Identify Improvement Path 	<ul style="list-style-type: none"> • Monitor Compliance • Document Level of Compliance 	<ul style="list-style-type: none"> • Verify Criteria Compliance • Provide Criteria Guidance
When Evaluated	<ul style="list-style-type: none"> • Assessment Before Development • Self-Monitor During Development 	<ul style="list-style-type: none"> • Throughout Development 	<ul style="list-style-type: none"> • Post Design • Primarily Post Development
How Evaluation is Performed	<ul style="list-style-type: none"> • Maturity questionnaire • On-site interviews with personnel • Review of documented procedures • Examination of capability evidence 	<ul style="list-style-type: none"> • Detailed on-site assessments of: <ul style="list-style-type: none"> - process evidence - process documentation - environmental controls 	<ul style="list-style-type: none"> • On/Off-site design analysis • Off-site: <ul style="list-style-type: none"> - review of assurance evidence - technical review boards (TRB) which perform independent QA • On-site testing
Who Performs Evaluation	<ul style="list-style-type: none"> • SEI or Government personnel • Organization does self assessments 	<ul style="list-style-type: none"> • Government personnel • Government representative 	<ul style="list-style-type: none"> • NSA personnel • NSA representatives
Result of Evaluation	<ul style="list-style-type: none"> • CMM level identified • Process strengths and weaknesses • Recommended improvements 	<ul style="list-style-type: none"> • Degree of compliance is documented 	<ul style="list-style-type: none"> • Given rating for highest TCSEC level where all requirements are satisfied • Final evaluation report (FER) which describes product and how it meets its requirements • Evaluated products list entry

Table 3: Description of Development and Evaluation Methods (Part 2)

	ITSEC/ITSEM	Federal Criteria³	CTCPEC
Introduction Date	1990	1992 (Draft)	1993
Purpose of Method	<ul style="list-style-type: none"> • Provide: <ul style="list-style-type: none"> - Guidance for developing - Metric for evaluating - Basis for specifying COTS trusted products & systems 	<ul style="list-style-type: none"> • Provide: <ul style="list-style-type: none"> - Guidance for developing - Metric for evaluating - Basis for specifying COTS trusted products 	<ul style="list-style-type: none"> • Provide: <ul style="list-style-type: none"> - Guidance for developing - Metric for evaluating - Basis for specifying COTS trusted products
Intended Target of Method	<ul style="list-style-type: none"> • COTS Trusted Products & Systems - for Government and Commercial Environments 	<ul style="list-style-type: none"> • COTS Trusted Products 	<ul style="list-style-type: none"> • COTS Trusted Products - for Government Environments
Premise of Method	<ul style="list-style-type: none"> • A common criteria will: <ul style="list-style-type: none"> - make trusted products more widely available - provide a basis for: measuring and specifying assurance requirements • Complete Freedom in: <ul style="list-style-type: none"> - feature selection will promote products which meet market needs 	<ul style="list-style-type: none"> • A common criteria will: <ul style="list-style-type: none"> - make trusted products more widely available - provide a basis for: measuring and specifying assurance requirements • Complete Freedom in: <ul style="list-style-type: none"> - feature - assurance selection will promote products which meet market needs 	<ul style="list-style-type: none"> • A common criteria will: <ul style="list-style-type: none"> - make trusted products more widely available - provide a basis for: measuring and specifying assurance requirements • Complete Freedom in: <ul style="list-style-type: none"> - feature selection will promote products which meet market needs
Strategy of Method	<ul style="list-style-type: none"> • Define specific assurance levels • Target of Evaluation (TOE) defines: <ul style="list-style-type: none"> - combinations of security features - a specific assurance level • Vendors can define their own TOE • Independent evaluation ensures compliance with selected security features and assurance level • Include effectiveness of security mechanisms in evaluation 	<ul style="list-style-type: none"> • Protection Profiles define combinations of: <ul style="list-style-type: none"> - security feature - assurance development - assurance measurement requirements • Vendors can define their own Protection Profiles • Protection Profiles can be targeted to meet particular needs • Independent evaluation ensures compliance with profile 	<ul style="list-style-type: none"> • Define specific security feature components <ul style="list-style-type: none"> - Allow vendors to select and build any desired combinations of these feature components • Define specific assurance levels • Independent evaluation ensures compliance with selected security feature components and assurance level
Organization of Criteria	<ul style="list-style-type: none"> • 3 Independent Areas <ul style="list-style-type: none"> - 6 Hierarchical Assurance levels E1 through E6 - 2 aspects of effectiveness - Vendor defined security features 	<ul style="list-style-type: none"> • 3 Independent areas define unbundled: <ul style="list-style-type: none"> - security feature - assurance development - assurance measurement components • Only predefined components can be used to form a profile 	<ul style="list-style-type: none"> • 8 Hierarchical Levels <ul style="list-style-type: none"> - T0 through T7 of - increased assurance • Unbundled security feature components
Requirements	How Evaluated	How Evaluated	How Evaluated
-Target Arch	Analyze	Analyze	Analyze
-Target Design	Analyze	Analyze	Analyze
-Target Implem	Test	Test	Test
-Assur Evid	Analyze	Analyze	Analyze
-Eng Proc		Confirm Process	Confirm Process
-Eng Envir	Confirm Controls	Confirm Controls	Confirm Controls
-Target Envir		Consider Controls Specified in Profile	
-Risk of Use			
Purpose of Evaluation	<ul style="list-style-type: none"> • Verify Compliance with Target of Evaluation 	<ul style="list-style-type: none"> • Verify Compliance with Profile 	<ul style="list-style-type: none"> • Verify Compliance with assurance level and selected security features • Assistance in development
When Evaluated	<ul style="list-style-type: none"> • Post Design 	<ul style="list-style-type: none"> • TBD by Criteria Authors 	<ul style="list-style-type: none"> • Post Design • Primarily Post Development
How Evaluation is Performed	<ul style="list-style-type: none"> • Off-site: <ul style="list-style-type: none"> - assurance documentation review - product documentation review - testing • On-site review and testing 	<ul style="list-style-type: none"> • TBD by Criteria Authors 	<ul style="list-style-type: none"> • Off-site: <ul style="list-style-type: none"> - review of assurance evidence - technical review boards • On-site testing
Who Performs Evaluation	<ul style="list-style-type: none"> • Government Certified Commercial Company 	<ul style="list-style-type: none"> • TBD by Criteria Authors 	<ul style="list-style-type: none"> • Government personnel • Government representative
Result of Evaluation	<ul style="list-style-type: none"> • Pass / Fail for features and assurance level specified in TOE 	<ul style="list-style-type: none"> • Pass / Fail for features and assurances specified in profile 	<ul style="list-style-type: none"> • Pass / Fail for assurance level and selected security features targeted

Table 3: Description of Development and Evaluation Methods (Part 3)

	RAMP	ISO 9000 Series	CCEP
Introduction Date	1989	1991	about 1983
Purpose of Method	<ul style="list-style-type: none"> • Maintain the assurance of previously evaluated Trusted Products 	<ul style="list-style-type: none"> • Provide guidance for software quality assurance 	<ul style="list-style-type: none"> • Ensure adequacy of design and correctness of implementation for COMSEC devices.
Intended Target of Method	<ul style="list-style-type: none"> • Evaluated COTS Trusted Products - for DoD Environments 	<ul style="list-style-type: none"> • All Software 	<ul style="list-style-type: none"> • Government COMSEC Devices
Premise of Method	<ul style="list-style-type: none"> • A defined process for maintaining the evaluation rating will: <ul style="list-style-type: none"> - ensure that new versions of evaluated products are still TCSEC compliant - keep the list of evaluated products current 	<ul style="list-style-type: none"> • Guidance for software quality assurance will: <ul style="list-style-type: none"> - establish quality principles - promote international consensus - address customer expectations 	<ul style="list-style-type: none"> • Documenting all COMSEC: <ul style="list-style-type: none"> - knowledge - guidance • and using this guidance will: <ul style="list-style-type: none"> - improve assurance of COMSEC devices
Strategy of Method	<ul style="list-style-type: none"> • Prescribe controls on the process used to maintain the product • Require vendors to perform their own security analysis • Require training for vendor security analysts (VSAs) • Vendors defend why the changes made maintain the rating from the previously evaluated product 	<ul style="list-style-type: none"> • Describe controls and methods for preventing nonconformance with requirements at all stages from development through maintenance • Require supplier to demonstrate capability to: <ul style="list-style-type: none"> - develop - supply - maintain software products 	<ul style="list-style-type: none"> • Establish Principles for: <ul style="list-style-type: none"> - Design - Development Environment • Select and tailor to each project • Analyze Design for: <ul style="list-style-type: none"> - Principle adherence - Single fault analysis
Organization of Criteria	<ul style="list-style-type: none"> • By Role: <ul style="list-style-type: none"> - Defines specific activities that must be performed by each role • All TCSEC requirements from the original evaluation still apply 	<ul style="list-style-type: none"> • By Life-Cycle Activities <ul style="list-style-type: none"> - while intended to be independent of the life-cycle model used, the organization is based on the fact that quality-related activities should be organized according to the life-cycle model used. 	<ul style="list-style-type: none"> • Compendium of principles for: <ul style="list-style-type: none"> - Design - Development Environment • Compendium <ul style="list-style-type: none"> - Ordered by topic - Not broken into levels
Requirements	How Evaluated	How Evaluated	How Evaluated
-Target Arch	Analyze Changes		
-Target Design	Analyze Changes		Analyze
-Target Implem	Test		Test
-Assur Evid	Analyze Changes		Analyze
-Eng Proc	Confirm Existence and Audit	Confirm Existence	
-Eng Envir		Confirm Controls	
-Target Envir			
-Risk of Use			
Purpose of Evaluation	<ul style="list-style-type: none"> • Ensure rating of product was maintained for new release 	<ul style="list-style-type: none"> • Validate supplier's capability to develop, supply, and maintain software products 	<ul style="list-style-type: none"> • Verify Principles Compliance
When Evaluated	<ul style="list-style-type: none"> • During and After Maintenance 	<ul style="list-style-type: none"> • Assessment before development 	<ul style="list-style-type: none"> • Throughout development • Emphasis on detailed design
How Evaluation is Performed	<ul style="list-style-type: none"> • Vendor performs their own security analysis of changes • Vendor defends analysis results • Government audits process 	<ul style="list-style-type: none"> • On-Site Audit of quality assurance activity evidence, e.g., during: <ul style="list-style-type: none"> - Planning - Design and Implementation - Testing and Validation 	<ul style="list-style-type: none"> • Detailed Evaluation <ul style="list-style-type: none"> - Review of developer provided analysis - Additional independent analysis
Who Performs Evaluation	<ul style="list-style-type: none"> • Vendor personnel (Must be NSA recognized VSA) • NSA personnel can as well 	<ul style="list-style-type: none"> • Registrars accredited by country's Registrar Accreditation Body (An industry supported entity) 	<ul style="list-style-type: none"> • NSA personnel only
Result of Evaluation	<ul style="list-style-type: none"> • Pass / Fail for target level • Ratings maintenance report and updated Final Evaluation Report • Updated evaluated products list entry 	<ul style="list-style-type: none"> • Recommended/Deferred/Not Recommended for compliance with the standard 	<ul style="list-style-type: none"> • Pass/Fail

Table 3: Description of Development and Evaluation Methods (Part 4)

	Profiling³	Certification	Accreditation
Introduction Date	1993	1960's	1960's
Purpose of Method	<ul style="list-style-type: none"> • Catalog and describe: <ul style="list-style-type: none"> - security products - systems 	<ul style="list-style-type: none"> • Ensure compliance with security requirements • Provide technical assessment to accreditor 	<ul style="list-style-type: none"> • Understand: <ul style="list-style-type: none"> - risk of fielding system - operational impacts due to vulnerabilities
Intended Target of Method	<ul style="list-style-type: none"> • Security Products or Systems <ul style="list-style-type: none"> - for DoD Environments 	<ul style="list-style-type: none"> • Government Systems 	<ul style="list-style-type: none"> • Government Systems
Premise of Method	<ul style="list-style-type: none"> • An independent assessment and documentation of: <ul style="list-style-type: none"> - security features - interoperability information of security products will: <ul style="list-style-type: none"> - provide useful information to acquisition organizations, certifiers, and accreditors 	<ul style="list-style-type: none"> • Independent technical analysis will: <ul style="list-style-type: none"> - ensure all requirements are met (including security) - identify strengths and weaknesses - support accreditation decision 	<ul style="list-style-type: none"> • An informed decision to operate will ensure security risks are: <ul style="list-style-type: none"> - understood - reduced to an acceptable level
Strategy of Method	<ul style="list-style-type: none"> • Does not levy any requirements • Examine vendor claims or program requirements • Examine products or systems to: <ul style="list-style-type: none"> - gather facts - test features - test interoperability - document results 	<ul style="list-style-type: none"> • Does not levy any requirements • Ensures that program's security requirements are met • Provides assessment information to accreditor 	<ul style="list-style-type: none"> • Does not levy any requirements • Identifies security risk • Ensures security features and assurance is sufficient to mitigate the security risk of fielding system
Organization of Criteria	• N/A	• N/A	• N/A
Requirements⁴	How Evaluated	How Evaluated	How Evaluated
-Target Arch		Review	
-Target Design		Review and analyze	
-Target Implem	Test	Review and analyze	Survey or Test
-Assur Evid	Review	Review	
-Eng Proc			
-Eng Envir			
-Target Envir	Considers for Systems	Review	
-Risk of Use	Identify		Identify and Understand
Purpose of Evaluation	<ul style="list-style-type: none"> • Identify Security Features • Investigate Interoperability • Develop pre-analyzed compositions of products 	<ul style="list-style-type: none"> • Ensure Requirements Compliance 	<ul style="list-style-type: none"> • Identify Risk of Use • Understand Risk • Make Accreditation Decision
When Evaluated	• Post Design	• Throughout Development	• Pre-Operation
How Evaluation is Performed	<ul style="list-style-type: none"> • Short Off-Site Assessment of product(s) in NSA or vetted industry lab 	<ul style="list-style-type: none"> • Assess requirements • Detailed assessment of system • Perform risk analysis • Analyze risk trade-offs 	<ul style="list-style-type: none"> • Site accreditation survey • Assessment of system risks and proposed countermeasures (certifier recommendation)
Who Performs Evaluation	<ul style="list-style-type: none"> • NSA personnel • NSA representatives (including industry) 	<ul style="list-style-type: none"> • Government personnel • Government representatives 	<ul style="list-style-type: none"> • Designated Approving Authority
Result of Evaluation	<ul style="list-style-type: none"> • Description of product or system • Risk of use report 	<ul style="list-style-type: none"> • Assessment of adequacy of system achieving mission • Certification recommendation to Accreditor 	<ul style="list-style-type: none"> • Decision to operate

Results

Based on the descriptions in Table 3, we present the following results, which include some trends, identify the primary assurance characteristics of these development and

³ Note that the description of the Evaluation Process for the Federal Criteria and Profiling efforts are estimates as evaluations following either of these methods have not yet been performed.

⁴ Note that the methods listed in part 4 of this table do not levy their own requirements. Rather, they ensure or confirm that the vendor claims or program requirements are actually met.

evaluation methods, and describe some potential relationships between them. We then present the SE CMM approach to assurance, which is based on these results.

Trends

In comparing the different development and evaluation methods reflected in Table 3, a number of trends become evident. First of note is the movement away from bundling together features to assurance, which can be seen in how the criteria are organized. The Federal Criteria extends this to the point where each desired feature and assurance technique can be chosen by the vendor. Secondly, the Government is no longer the lone driver of these efforts, which is illustrated by who performs evaluations. More and more, industry is also driving them or, as is the case of ISO 9000, taking the lead. Lastly, there appears to be a trend to recognize that one must truly affect the engineering in order to best affect the end product or system (e.g., CMM, TSM, ISO 9000). Overall these trends reflect a shift from rigid government controlled methods to more flexible cooperative ones.

Assurance Characteristics

In order to differentiate the assurance aspects of these development and evaluation methods, we identified their key assurance characteristics. These are based on the following observations:

- Assurance activities are normally divided between the *production* of the assurance evidence and the *evaluation* of the evidence.
- The assurance benefits achieved by different methods varies.
- The explicit target of the method is sometimes not the same as the end target. For example, the CMM explicitly targets the software process but it is intended to ultimately affect the software produced.
- The assurance benefit varies with the explicit or end target. For example, the TSM tightly controls the software development process, which ultimately should minimize the number of errors in the software produced.

From these observations, we identified the following primary assurance characteristics of a development and evaluation method:

- Assurance method, in terms of both:
 - Production method (specifically what is produced, how it is produced, who produces it, and when) and
 - Evaluation method (specifically how it is evaluated, who evaluates it, and when)
- Assurance benefits, in terms of both:
 - Direct benefit relative to the explicit target; and
 - Indirect benefit desired for the end target.
- Assurance target, in terms of both:
 - Explicit target; and
 - End target (a system or product), which may be the same.

Table 4 summarizes these assurance characteristics for each method we examined:

Table 4: Taxonomy of Assurance of Development and Evaluation Methods

	Assurance Method		Assurance Benefit		Assurance Target	
	Production	Evaluation	Direct	Indirect	Explicit	End
CMM	Guide Process Improvement	Assess Current Development Process	Improve Development Process	Improve Software quality	Engineering Process	All Software
TSM	Levy Strict Controls	Document degree of Compliance	Control Development Process	Minimize errors	Development Process and Environment	Software for Strategic Defense
TCSEC / ITSEC / Federal Criteria / CTCPEC	Require Specific Assurance Activities	Ensure Compliance with respective Criteria	Ensure adequate enforcement of respective security policy	N/A	Security Relevant Software and Hardware	<u>Same</u>
RAMP	Define Required Maintenance Process Elements Train VSAs	VSA Ensures Compliance Government audits process, reviews security analysis	Ensure adequate security analysis and product controls	Ensure adequate enforcement of security policy	Maintenance Process	Security Relevant Software and Hardware
ISO 9000 Series	Define Software QA Standard	Assess Compliance with Standard	Improve Software QA Process	Improves quality	Quality Assurance Process	All Software
CCEP	Define COMSEC Principles and Analysis Techniques	Ensure Compliance	Ensure correctness, tamperproofness, and single faults will not cause policy violations	N/A	Entire Device, including Software, Hardware, and Container	<u>Same</u>
Profiling	N/A	Assess	Verify vendor functionality & interoperability claims	N/A	Products and Systems	<u>Same</u>
Certification	N/A	Requirements Compliance Verification	Ensure requirements are met and system can accomplish mission	N/A	Operational System	<u>Same</u>
Accreditation	N/A	Risk Analysis	Risks are mitigated, or understood and accepted	N/A	Operational System	<u>Same</u>

This table illustrates the differences in the type of assurance these methods provide. Much like intensity, hue, and tint define the dimensions of color, these three characteristics can be thought of as defining the 'color' of the assurance provided by the

method. Unlike for color, these assurance dimensions (i.e., method, benefit, target) are not completely independent of one another. Specifically, the assurance benefit is completely dependent on the method since it is the assurance method which defines the activities that are performed to produce the assurance.

The assurance target, however, is somewhat independent of the assurance method and benefit. Although the assurance method was developed to produce a specific type of assurance for a specific target, it is usually possible to apply the method to a different target with some interpretation. However, it is important to recognize the limits of these methods. First, the benefit to the target is defined by the assurance method and not the target. For example, applying the CMM to a trusted system may improve the development process but not necessarily provide sufficient assurance that the security policy is adequately enforced. Second, the benefit gained may not make sense for the specific target. For example, applying the TCSEC to a non-trusted system is nonsensical, since a non-trusted system has no security policy to enforce.

Relationships Between Methods

Although Table 4 differentiates the assurance aspects of these development and evaluation methods, there are potential relationships between them. First, for two methods with the same end target, it is possible to combine and/or trade-off approaches. For example, if it is known that a more mature process was used to design, develop, and test the implementation, then one could reasonably consider reducing the reliance on analysis of the end product to gain additional assurance. Second, for those methods with end targets that may be composed of several products, the assurance gained by the product development methods compliment the assurance gained by the method used to compose the products into a system. This second relationship is the notion behind the various trusted product evaluation criteria, which use product evaluation results to support the development of systems composed of these products.

SE CMM Assurance Approach

The objective of the SE CMM is to help security engineering organizations define and improve their security engineering process. This paper has illustrated the diversity of current development and evaluation methods that produce assurance. Any one or more of these methods may be required as part of a given project. Therefore, an organizational process must be adaptable to these diverse methods. The SE CMM promotes such a process by introducing practices that help an organization define and improve their ability to efficiently and effectively produce assurance, regardless of the assurance method. SE CMM practices help an organization incorporate any specific assurance requirements, activities, or roles required for a specific project. By recognizing this incorporation of specific assurance requirements as part of the organizational process, the SE CMM practices adapt to and strengthen any development and evaluation method required by a specific project.

This SE CMM approach improves assurance both directly and indirectly. First, assurance is directly improved by applying a sound engineering process to project assurance activities (e.g., efficiently and effectively incorporating project specific

assurance requirements into a defined organizational process and ensuring coordination between activities). Second, assurance is indirectly improved through practices within the security engineering process which focus on improving the entire security engineering process (e.g., process definition, measurement, verification, and improvement), which will in turn improve the product.

Conclusions

While often discussed as being different ways for achieving the same result, the assurance gained from various development and evaluation methodologies is actually quite different. Just as colors can vary in intensity, hue, and tint, the assurance provided by these methodologies can vary in several dimensions (i.e., method, benefit, target) as well. To reason about the assurance to be gained from a particular methodology, these characteristics must be examined in detail and in ways that distinguish among and between the production and evaluation method, the direct and indirect benefit, and the explicit and end target.

Some form of assurance has become a standard requirement in the development of computer products and systems. Existing and emerging development and evaluation methods offer a myriad of techniques to produce and measure the assurance required. The community needs to understand and be able to recognize the differences in the assurance provided by various development and evaluation methods. In addition, future work needs to provide a more detailed taxonomy of assurance production and evaluation techniques. Such a taxonomy would help the community understand and focus on issues and support trade-off analyses when developing new methodologies, combining existing ones, or identifying the assurance activities necessary for specific products or programs.

References

- [1] *Webster's New World Dictionary, 3rd College Edition*, Prentice Hall, 1991
- [2] Richard N. Bolles, "What Color is Your Parachute?" Ten Speed Press, Berkeley, CA, 1993
- [3] Karen M. Ferraiolo, Jeffrey R. Williams, Douglas J. Landoll, "A Capability Maturity Model for Security Engineering," Proceedings of the 1994 Canadian Computer Security Conference, May 1994
- [4] Software Engineering Institute, *Capability Maturity Model for Software*, Version 1.1, February 1993

Bibliography

Canadian Trusted Computer Product Evaluation Criteria, Version 3.0e, Canadian System Security Centre, Communications Security Establishment, January 1993

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

Department of Defense, *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 (Also known as the 'Yellow Book')

Information Technology Security Evaluation Criteria, Harmonised Criteria of France - Germany - the Netherlands - the United Kingdom, Version 1.2, June 1991

Information Technology Security Evaluation Manual, April, 1992

International Organization for Standardization, *Quality Management and Quality Assurance Standards - Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software*, ISO 9000-3, 1991

National Institute of Standards and Technology, *Federal Criteria for Information Technology*, Version 1.0 (DRAFT), January 1993

National Security Agency, *Specification for General Functional Security Requirements for a Telecommunications System* (SECRET), 2 June 1991

National Security Agency, *Rating Maintenance Phase Program Document* (DRAFT), Version 2.0, October 1993

Strategic Defense Initiative Organization (Now the Ballistic Missile Defense Organization), *Trusted Software Methodology*, June 1992

BFЕ Applicability to LAN Environments

17th National Computer Security Conference

Abstract: BLACKER Front Ends (BFЕs) were originally designed for use in X.25 packet switch networks. Today BFЕs are the only packet encryption device with an A1 level of trust. However, network topologies currently rely more on Local Area Network (LAN) infrastructures than X.25, potentially limiting the use of BFЕs as security devices. This paper documents testing performed to determine if BFЕs can be easily integrated into LAN environments. BFЕs enhanced to support higher serial clock rates form an important component of the testing.

The testing involves the insertion of BFЕs into an existing single-level operational internet. The BFЕs are used to protect (via encryption) test data while it traverses the operational internet. BFЕs are already in operational use on this network in their traditional X.25 connection mode, so it is also necessary to maintain cryptographic separation between the operational and test BFЕ traffic.

The testing demonstrates that BFЕs can be integrated into LAN environments to provide security services to the LAN-attached systems (as well as Wide Area Network (WAN) usage). The throughput capabilities are adequate for many of the applications commonly used.

Authors:	Tom Benkart	Dave Bitzer
	Director of Engineering	Member of Technical Staff
	ACC Network Systems	DoD
	Voice: 410-290-8100	Voice: 301-688-6058
	FAX: 410-290-8106	
	teb@sys.acc.com	Bitzer@DOCKMASTER.NCSC.MIL

Government sponsor:	Security Proof Of Concept Keystone Program
	DoD/Keith Abernethy
	Contract MDA904-93-C-G090

Keywords: Access Control Center, ACC, BLACKER Front End, BFЕ, Discretionary Access Control, DAC, Key Distribution Center, KDC, Mandatory Access Control, MAC, Multi-Level Secure, MLS, Network Security, Packet Security, Security Architecture, Security Policy

Introduction

BLACKER Front Ends (BFEs) were originally designed for use in X.25 packet switch networks. Today BFEs are the only packet encryption device with an A1 level of trust. However, network topologies currently rely more on Local Area Network (LAN) infrastructures than X.25, potentially limiting the use of BFEs as security devices. This paper documents testing performed to determine if BFEs can be easily integrated into LAN environments. BFEs enhanced to support higher serial clock rates form an important component of the testing.

The testing involves the insertion of BFEs into an existing single-level operational internet. The BFEs are used to protect (via encryption) test data while it traverses the operational internet. BFEs are already in operational use on this network in their traditional X.25 connection mode, so it is also necessary to maintain cryptographic separation between the operational and test BFE traffic.

BLACKER Overview

The BLACKER system is a COMSEC system for packetized data that comprises three devices: a BFE, a Key Distribution Center (KDC), and an Access Control Center (ACC). It is approved for encryption of all levels of classified traffic. It also meets all the requirements of a COMPUSEC system at the A1 level of trust.

As its name implies, the BFE is designed to be placed at the front end (network side) of a single host or an IP router front-ending a collection of hosts. The generic term "site" is used to refer to either of these configurations. The site could be single level (untrusted) or Multi-Level Secure (MLS). The BFE will protect that site from malicious external networks and preserve the trustedness of a site-to-site connection across an untrusted network. BFEs provide both mandatory and discretionary access control.

The KDC is an automated generator and distributor of all the encryption keys for a group of BFEs. The KDC uses the packet data network itself to securely distribute the needed keys to each BFE. An ACC is paired with each KDC to provide the access tables, permissions, audit functions, and other system status and control features needed to support the system.

BFEs have been operational since 1989. The usage has been primarily with X.25 packet switch networks (with access link rates constrained to 64 Kbps and below) and single-level hosts. This testing utilizes enhanced BFEs capable of supporting X.25 interfaces operating at clock rates of 400 Kbps, as well as prototype Ethernet BFEs with Ethernet interfaces on their network (black) side.

Integrating BFEs into LAN Environments

Integrating current BFEs into LANs requires protocol conversion between X.25 and the LAN protocol (Ethernet in this testing). The protocol conversion is accomplished in two ways in this testing. First, internal conversion using prototype Ethernet BFEs. Second, COTS IP routers with both Ethernet and X.25

interfaces can be used on either side of the BFEs. With either approach, the X.25 interfaces of the BFEs are transparent to the remainder of the LAN infrastructure.

Since packet switches are not involved in these configurations, the clock rates of the X.25 serial lines are no longer constrained to the 64 Kbps range. This testing uses existing BFEs at interface clock rates up to 150 Kbps and enhanced BFEs at rates up to 400 Kbps.

Although using IP routers as protocol converters might appear to significantly increase the cost of using BFEs, the availability of low-cost routers makes this approach very attractive since it does not require any changes to or reevaluation of the BFEs. Note that the cost of a BFE and routers is less than the cost of one comparable Motorola Network Encryption System (NES).

Security Policy

The security policy for this testing requires the same, or better, access control to sites and individual hosts within sites than is now provided on the operational network. The policy comprises two parts. The mandatory part states that all personnel will have access only to those systems for which they hold formal clearances. The discretionary part states that clearance alone is insufficient justification for access. Individual permissions shall be verified and mediated for test access. The applicable laws, regulations, and security requirements stated below constitute the security policy.

Mandatory component: No host will be connected to another host at other than a common security level, that level being set by the system security administrator. A host is defined as any device, computer, router, bridge, etc. containing data at a security level or levels, or containing a process that when invoked may divulge information at a security level. "Connected" means any virtual or physical circuit, intended or unintended, or the ability to invoke remote processes, or pass traffic on that circuit. "Common level" means read and write access at that level at both hosts. Hosts may be untrusted or trusted, and the National Computer Security Center (NCSC) yellow book will be followed with respect to range of connectivity.

Discretionary component: No host will be connected to another host without the direct permission of the test System Administrator. Strict configuration control will be maintained and no unnecessary connectivity will be allowed. Cryptographic isolation from all operational traffic, including operational BLACKER traffic, will be mandated.

Architectural Approach

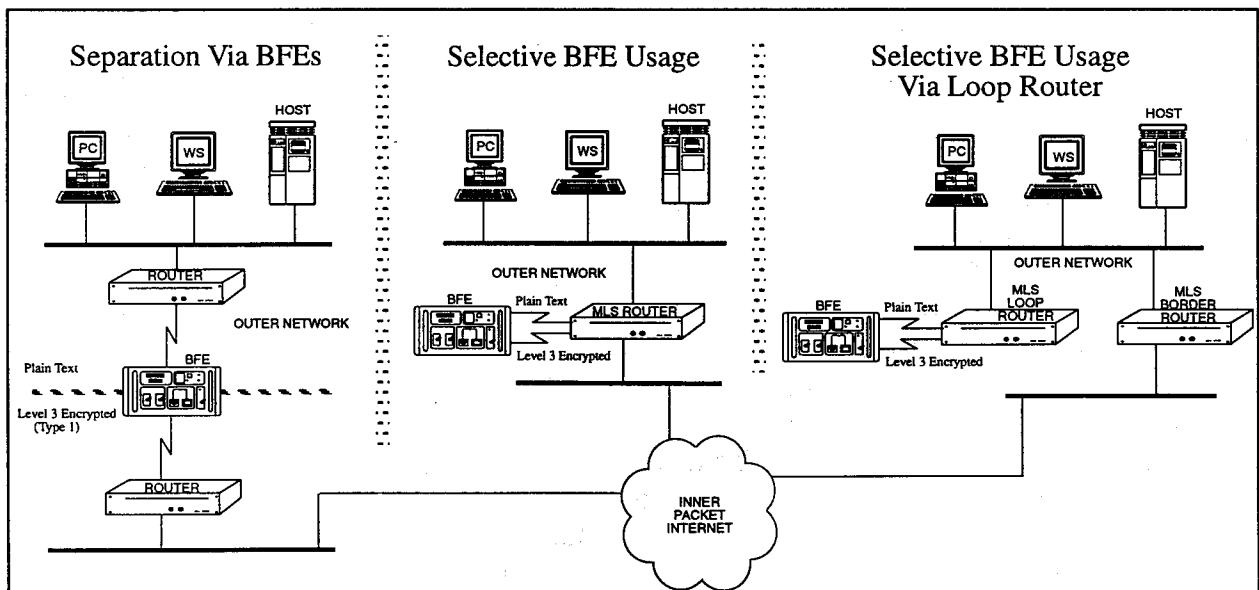
The security architecture calls for viewing the existing system-high internet as an inner (backbone) network, and surrounding that network is a second, potentially MLS, outer internet. BFEs and routers provide integrity and confidentiality between the networks, so that:

1. The outer network(s) uses the inner inter-network for transport services and views the inner inter-network as a class A network (e.g., network

21.0.0.0). The BFEs and routers provide IP-level security services connecting the outer and inner networks.

2. The inner inter-network views these devices as hosts or routers on their network and has no view of the outer network(s). If the inner and outer networks have a security level in common, limited connectivity may be permitted at that common level.
3. The BFEs and routers enforce both the confidentiality and integrity policy requirements. The integrity policy preserves host and level of origin assurance and includes authentication and protection against spoofing and modification by systems within the inner network. Crypto key separation is based on both host pair and security/compartments level of the data being transmitted. Electronic key distribution counters handling problems and formal system certification provides high assurance of continual policy enforcement, trusted distribution, and trusted recovery as specified by 5200.28-STD criteria at the A1 level.

This test involves three distinct implementations of the security architecture. One requires all traffic to pass through the BFEs, while the others support selective usage of the BFEs. All three implementations are depicted in the following figure.



Separation Via BFEs

With this implementation, all traffic leaving a site is encrypted while traversing a backbone packet network of a different (either higher or lower) level. The BFEs are integrated into the LAN environment by connecting them to routers that support both Ethernet and X.25 interfaces. In order to communicate through a BFE, two entities must be operating at the same level. Since BFEs are MLS devices, a single BFE can support simultaneous connections to multiple entities of varying sensitivity levels on its host side.

The BLACKER System Administrator has full control of access authorizations between sites, at an A1 level of trust. Confidentiality is assured across the inner network and integrity is assured in that the inner network hosts or processes cannot initiate or alter a valid message. Because of the MLS nature of the BFE, it becomes easy to add or change a security level or compartment, as opposed to single-level cryptographic devices that must be replicated at each level/compartment.

This implementation adheres stringently to the integrity and confidentiality requirements defined above. No connectivity between the inner and outer networks is permitted. The BFEs can further limit connectivity between sites via DAC mechanisms. DAC mechanisms within the routers can provide finer granularity on connectivity between inter-site hosts. If a single level of data is involved, the router need not be MLS. If multiple levels are involved, the router must be MLS. Individual hosts could still be single level if they are isolated to LAN segments of a single level (connected to the MLS router).

Selective BFE Usage

With this implementation, not all traffic entering or leaving a site passes through a BFE. Instead, all traffic passes through a trusted router and the router controls which traffic passes through the BFE. For example, traffic destined to a particular remote site will always have a known destination IP address. The routing tables can be configured to automatically forward all datagrams for that site to the BFE for encryption. The decision to forward the datagram through the BFE can be based on other information as well, such as an IPSO label. In this implementation, connectivity between sites is controlled by the network administrator via configuration of the router, as well as the BLACKER system. Note that the BFE plays a second key role here by providing the network administrator a trusted channel to remotely control the configuration in each router, preventing spoofing and unauthorized changes.

This implementation inherently operates in a multilevel mode, since it supports simultaneous communication with remote systems at more than one level. Therefore, the router must be trusted. Since proper implementation of the security policy depends on proper labeling of the datagrams, any hosts communicating with inner systems must also be trusted (other hosts could be isolated to single-level LANs by the trusted router). Note that without encryption, confidentiality is lost for the traffic that leaves the site and integrity is not assured for incoming traffic. The level of trust for this implementation would be the lesser of the levels of trust of all the trusted systems.

Selective BFE Usage Via Loop Router

With this implementation, not all traffic entering or leaving a site passes through a BFE. Datagram flow for outgoing traffic is controlled by the end users by specifying the loop router or border router as their next hop (this can be done for all destinations or selectively for different destinations). If the loop router is

selected, it controls whether or not datagrams should be sent through the BFE before being forwarded to the border router. Incoming traffic is forwarded directly to the end system by the border router, if it is not encrypted, and is forwarded to the BFE (through the loop router) if it is encrypted. In this instance, connectivity, confidentiality, and integrity are controlled by the users via the routing tables on the (trusted) end systems, rather than the network administrators. Note that the BFE again provides the network administrator a trusted channel to remotely control the configuration in each router, preventing spoofing and unauthorized changes.

For MLS operation, this implementation requires trust in all systems within the site. It is especially suited for a transition strategy while this architecture is being implemented on an existing (system-high) internet, since it can be transparent to any existing hosts and routers.

To take advantage of the confidentiality and integrity services provided by the BFEs, a prudent network administrator could use the Selective Usage or Loop Router implementations even when the security levels of the inner and outer networks are the same.

Summary

All of the implementations are interoperable if permitted by the security policy. Each one offers different capabilities to the site hosts, with corresponding requirements of the level of trust placed in those hosts. Assurance levels are specified by DoD Directive 5200.28.

Test Description

In order to provide "real-world" results, all the testing was performed across an existing operational internet. The goals were to demonstrate the feasibility of BFE usage on LANs, measure the performance capabilities of BFEs in that role, and demonstrate that the security policy was correctly enforced.

Testing began with simple configurations to gain experience with the security architecture. This phase included usage of the prototype Ethernet BFEs. At that time, no users were dependent on the BFEs for any operational requirements. This phase lasted just long enough to gain confidence in the performance, reliability, and security of the systems.

The second phase involved operational usage of the BFEs by a limited number of users. This usage focused on typical day-to-day requirements such as sending and receiving email, remote host access (e.g., telnet), and file transfers. Since the existing network infrastructure could not be disturbed, the Loop Configuration described above was most commonly used. This implementation permitted individual users on a LAN to use the BFE simply by changing their default router to be the Loop Router. No other users on the LAN were affected. The Loop Routers were usually configured to insert and strip the IP Security Option (IPSO) labels required for MLS operation. Since high-trust routers and hosts were not available for the testing, the range of levels was severely

constrained. However, the concepts for greater separation with high-trust systems were demonstrated.

The final test phase added additional participants. This was significant since the added users did not have detailed knowledge of BFEs or network routing, requiring the security services to be totally transparent. In all, over 30 users at 24 sites were active participants. They were encouraged to use their systems in a normal manner and report any perceptions of added delay introduced by the BFEs. This phase lasted several months.

Performance Measurements

Quantitative measures of the overall system performance were obtained via three different mechanisms: a throughput measurement tool, ping, and FTP.

The throughput measurement tool was used in two different configurations. First, two BFEs connected via a single Ethernet segment (with no other traffic present on the LAN) were used in determining the maximum throughput of the BFEs. The throughput measured in this configuration was 266 Kbps for bidirectional traffic and 297 Kbps for unidirectional traffic. Second, throughput was measured across the internet both with and without BFEs in the communications path. Throughputs were measured at 48 packets per second with BFEs and 91 packets per second without BFEs.

Ping proved invaluable as a network debugging tool as a simple and reliable indicator of network connectivity. The round trip times (RTT) reported by ping were very consistent and were measured both with and without BFEs in the communications path. Across the internet, RTTs were consistently 25 to 40 ms without BFEs and 160 to 270 ms with BFEs. The times reported were sensitive enough to indicate whether the BFEs in the path were running at the higher clock rates. Since BFEs are known to require approximately 25 ms to process and forward a datagram, the measured values are consistent with the expected results.

FTP was the most extensively used performance measurement tool, since it reflects usage that is representative of most operational users. FTPs were performed between many different system pairs with a wide range of variables. These variables include:

Maximum Transmission Unit (MTU) - Each host is responsible for choosing the datagram size for messages it creates. This value is known as the MTU. Since the traffic flow involves transmission between systems on different networks, the default MTU is fixed at 576 bytes regardless of the actual optimal size. Systems which implement Path MTU Discovery are able to dynamically discover the actual optimum message size.

Clock Speed - The serial clock speed on the X.25 interfaces between the BFEs and routers is a limiting factor. The clock speed may also influence BFE behavior such as flow control and retransmissions between it and the router.

Receive Window - The recipient of the file transfer paces the sender by permitting only a specified number of bytes to be in transit at any time.

The ideal setting of this parameter results in the sender exhausting its window just as the acknowledgements are received for the first bytes of data in the window. Since BFE processing increases the round trip time (RTT), the acknowledgements take longer to be received and the ideal window size is larger than without BFEs.

Number of Large Buffers - The PC/TCP software used on PCs includes a variable for the number of large buffers allocated for datagram transmission. When the receive window was increased, it was necessary to increase this parameter to take full advantage of the larger window.

It quickly became apparent that the operational traffic on the internet did significantly impact the throughput measurements achievable. The internet was not a transparent pipe of "infinite" capacity, but a real system (frequently heavily loaded) that did impact the traffic flow of the test systems. The best throughput numbers were invariably obtained during off hours when the internet was least heavily used.

The results indicate that the single most important variables are the receive window size and number of large buffers (applicable to PC/TCP only). The BFEs add processing delay to the overall communications path and maximum throughput can be obtained only when the sending system never stops transmitting. On UNIX systems the default values for these variables are normally high enough to work quite well with the BFEs in the path. However, the default values for the PC/TCP software resulted in poor throughput.

The highest throughput measurement obtained was 170 Kbps. This was not made in a sterile laboratory environment, but over the operational network. The "typical" rate was approximately 80 Kbps. This value is comparable to the results measured on this network without the BFEs in the path. Perhaps the most significant observation to be made is that the FTPs did complete regardless of the throughput, proving the ability of the TCP implementations to dynamically adjust to the transfer rate available from the network at any time.

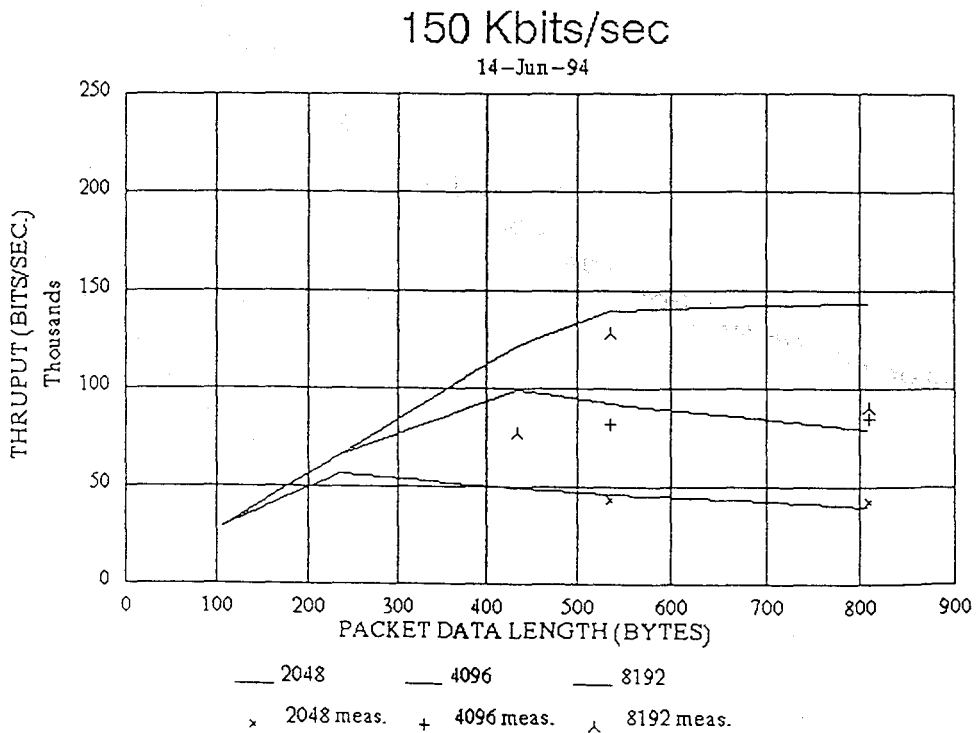
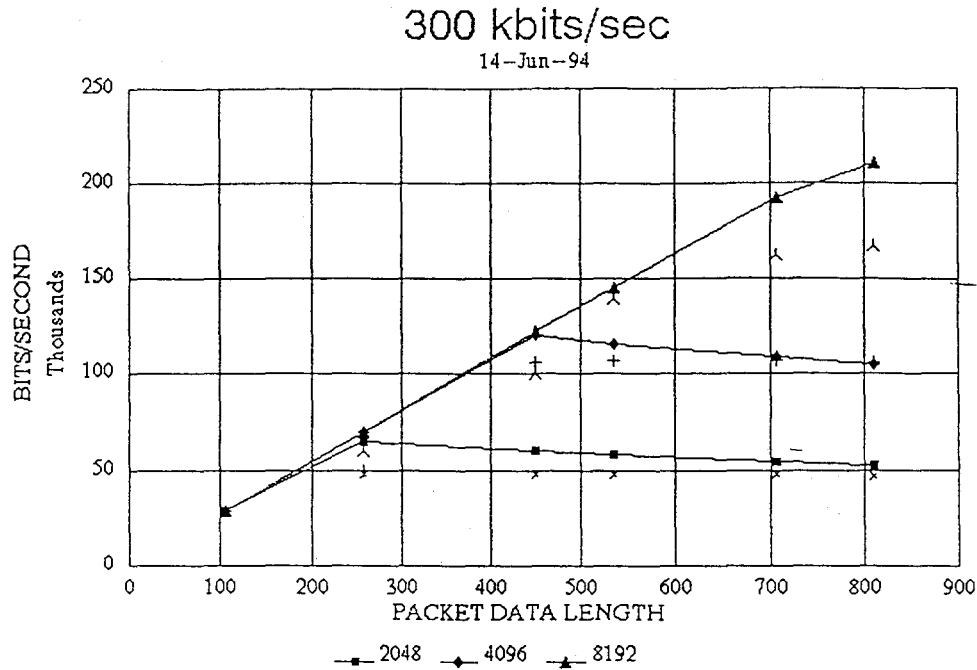
Performance Model

During the test, personnel developed a performance modeling tool for the system. The model has as parameters the bit rate (clock speed), packet data length (MTU), fixed processing delay (per packet), FTP delay, and TCP receive window size. It assumes a 40-byte datagram header and a 50-packets-per-second processing limit (by the BFEs). Throughput predictions are generated for receive window sizes of 2048, 4096, and 8192 bytes. The model assumes that sufficient transmit buffers are available in the transmitting system never to throttle datagram transmission, no message loss, and no retransmissions. Sample graphical output from the model is shown below for interface clock rates of 150 and 300 Kbps.

The model points out several interesting factors. Before analyzing the output from the model, one assumption made by the tester was that throughput would improve by using larger packet sizes (in FTP transfers). As the graphs show, this is not necessarily the case. As packet sizes increase, the length of time

required for acknowledgements to reach the sending system increased. This could result in the transmit window closing, resulting in an overall drop in throughput. This factor can be alleviated by using larger TCP windows.

The testers also assumed that simply increasing the clock speed of the BFE interfaces would result in significantly better throughputs. However, by comparing the two graphs it is obvious that the clock speeds become significant only when large TCP windows are used.



Conclusions

1. BFEs can be successfully integrated into LAN/router environments.
2. All three implementations of the security architecture are functionally sound.
3. The test participants stated that the throughput available from the three implementations is adequate for many types of their operational traffic, including email, remote login (telnet), and file transfers (FTP).
4. Tuning of parameters in the end systems is important for obtaining optimum throughput.
5. Sites can be quickly converted to the implementations described in this paper because the BFE infrastructure (i.e., access control and key management policies and procedures, ACC and KDC) already exists.
6. The prototype Ethernet BFEs operate as intended.
7. Although there are few trusted hosts on the Evaluated Products List (EPL) today, the BFE is ready to provide full network security services to MLS hosts on LANs in accordance with the DoD Directive 5200.28, as they become available.
8. The lack of widespread availability of high-trust hosts and (especially) routers limits the range of levels which can be processed by this architecture.

Bibliography

BLACKER Front End LAN Security Keystone, ACC Network Systems, December 1993.

CX/SX Trusted Facility Manual, Harris Computer Systems, May 1992.

DoD Directive 5200.28, Security Requirements for Automatic Data Processing (ADP) Systems, revised April 1978.

Filter Router Test Report Executive Summary, ACC Network Systems, April 1994.

Interface Control Document Including ICD Supplement for BLACKER Front End, BLACKER Program Phases 2 and 3, July 1993.

National Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, December 1985.

National Computer Security Center, *Final Design Assessment Report of Ford Aerospace Corporation Multinet Gateway System Advanced Development Model (Version 4.0)*, C22-REPT-01-90, June 1989.

National Computer Security Center, *Trusted Network Interpretation*, NCSC-TG-005, July 1987.

PC/TCP Network Software for DOS, FTP Software, February 1993.

The Architecture of Triad: A Distributed, Real-Time, Trusted System

Authors:

E. John Sebes
Nancy Kelem

Trusted Information Systems, Inc.
444 Castro Street, Ste 800
Mountain View, CA 94041
415/962-8885

Terry C. Vickers Benzel
Mary Bernstein Eve Cohen
Jeff Jones Jon King
Trusted Information Systems, Inc.
11340 W. Olympic Blvd., Ste 265
Los Angeles, CA 90064
310/477-5828

Michael Barnett
David M. Gallon
Roman Zacjew
Locus Computing Corporation
5910 Pacific Center Blvd.
San Diego, CA 92121
619/546-9500

Abstract

The Triad project is a prototype trusted operating system development. The name Triad represents the trio of requirements which this system must satisfy: multi-level security, real-time processing, and distributed processing. The goal of this project is to merge and advance the research in these three areas. This paper describes the background, design approach and trade-offs, features, and architecture of the Triad system.

Keywords: Trust, real-time, distributed systems, Mach, B3, thread migration, distributed IPC, scheduling coherence.¹

1 Overview

The Triad project is a prototype trusted operating system development, named for the trio of requirements which this system must satisfy: multi-level security (MLS), real-time, and distributed processing. The goal of this project is to merge and advance the research in these three areas, with a focus on providing processing capabilities for real-time distributed military C³I applications which process information of different classifications.

The motivation for Triad development is the observation that it is easier to add real-time and distributed processing functionality to a MLS operating system (OS) than it is to add security to a real-time and/or distributed system. Therefore, Triad will be developed by augmenting an existing trusted OS base with features that support real-time distributed processing. The OS base, TMach² [1], has been designed to be extensible, portable, and supportive of distributed processing. This base system is being augmented with real-time and distributed processing features, including: adaptable real-time scheduling within a replaceable scheduler framework (described in Section 5.2); scheduling coherence, shuttles, and passive servers (described in Section 5.3); distributed inter-process communication (described in Section 5.4); and replication of servers and objects (described in Section 5.5). These features will provide the basic abstractions and functionality identified in the previous study phase of the project [2].

The Triad System design emphasizes both reuse and assurance, by extending TMach's B3-targeted layered architecture to incorporate existing mechanisms from current research in real-time scheduling, distributed inter-process communication (IPC), and replication. The goal is to maintain the B3-level assurance of the base trusted OS, while adding the new functionality identified in a previous study phase of work based on the Alpha distributed real-time OS [3]. Trade-off analysis is also a key element of the design approach. When adding this new functionality, we must carefully weigh the effects of requirements from each of the three areas on requirements from the other two, in order to achieve an appropriate balance of functionality and resolve any conflicts.

The next section describes the background of the project, and its basic issues. Then, Section 3 describes our approach to addressing these issues, and Section 4 describes the trade-offs inherent in the issues. Finally, Section 5 describes the architecture of the Triad System, and the features provided within that architecture.

¹This project is supported by Rome Labs, under U.S. Government contract number F30602-93-C-0235.

²TMach is a Registered Trademark of Trusted Information Systems, Inc. (TIS)

2 Background

The real-time components of Triad represent the evolution of real-time systems that span the domain of complex, heterogeneous, industrial and military systems. In early real-time systems the design focus was sheer speed of execution, using minimal executives and careful application tuning, in order to synchronize with external world events. As processor power increased and costs dropped, more generalized approaches for designing and re-using real-time system software began to emerge. Modern real-time system designs take advantage of the techniques of computational predictability analysis and adaptable scheduling algorithms. The basic intent is to enable the completion of a function *at the right time* taking into account computation time, resource access time, and other factors, rather than simply striving to perform the function with as few instructions as possible.

This change of focus from *fast* to *predictable* allows a richer functional mix within a single real-time system. For example, it enables the incorporation of distribution technology which, in turn, enables the use of redundancy and replication of processing resources to increase a real-time system's survivability.

The need for multi-level security is becoming manifest in parallel with the increasing technical scope of real-time systems. The drive to integrate multiple applications brings with it the need to separate the applications algorithms and data from one another since they may embody drastically differing sensitivity levels and integrity levels. The traditional "really fast" real-time design pitted the use of trust mechanisms against mission success. However, the modern generalized real-time system design avoids that pitfall and largely mitigates many such design goal conflicts. In addition, the use of "system build" techniques [4] (described in Section 4.1.1) is used to further eliminate any remaining potential conflicts that can arise at run-time.

Distributed system requirements are an additional factor of the increasing scope of both real-time systems and MLS systems. A distributed system is a collection of computers connected by a high-speed interconnect or network. This collection, or cluster, is controlled by software that makes the underlying interconnect largely invisible. As a result, ordinary users and application software view the cluster as a single computer system, and can remain unaware of its distributed nature. Such distribution functionality is essential not only for geographically distributed command and control software, but also for real-time systems which require resource redundancy for fault tolerance.

The Triad Project is a research effort sponsored by Rome Labs, awarded August 1993, to produce a robust proof-of-concept prototype demonstration in December 1995. We will complete requirements definition stage in mid-1994 and the software detailed design in the second half of the year. In addition to the technical goals described here, the Triad project will employ an integrated 2167a development process which incorporates TCSEC security documentation and review with that of generic software development. The project stems from earlier Rome Lab sponsored research in the area of multi-level secure distributed operating systems (MLS DOS study) [2].

3 Approach

The Triad System will incorporate modern operating system features from the three basic areas, to form a TMach-based operating system supporting the basic functionality identified by the MLS DOS study effort, specifically its programming abstraction, scheduling algorithm, and formal security policy approach. The use of the programming abstraction (threads making distributed object-oriented remote procedure calls) is described in Section 5, and is facilitated by the addition to TMach of shuttles, described in Section 5.3. The scheduling algorithm usage is in the context of the replaceable scheduler interface work, described in Section 5.2. Finally, the Triad policy has already been formulated by merging elements of the MLS DOS policy and the TMach policy.

Various features of TMach are central to our approach. Most importantly, TMach provides the MLS functionality needed for Triad. Our decision to use TMach is based in part on the judgement that it is easier to add real-time and distributed processing functionality to a MLS system than it is to add security to a real-time and/or distributed system. Due to TMach's B3 layered architecture and modular implementation, it turns out that real-time and distribution extensions are required for a small number of components, and that the real-time extensions are largely orthogonal to the distribution extensions.

Likewise, since TMach is designed to be extensible, it lends itself well to the kind of extensions needed for the Triad system and Triad applications. Being composed of several distinct servers, TMach can be

readily augmented by new servers, such as application-level servers implementing new application types of objects. The object-oriented design of TMach server also means that they can be readily augmented with new subsystems to implement distribution. Further extensibility is derived from the Mach microkernel [5] which was designed to be extended into a distributed environment in the manner required for Triad. Thus, not only does extensibility at both the TMach server level and Mach microkernel level support the addition of distribution functionality, but this extensibility was explicitly designed partly for distribution.

Finally, because TMach is built on top of the Mach microkernel, it has portability benefits that are critical for distribution. The microkernel is structured so that all hardware-specific (non-portable) software is isolated, so as to permit straightforward modification for porting to multiple platforms. The TMach software itself has no hardware-specific portions. As a result, it can run on any hardware base to which a suitable version of the microkernel has been (or will be) ported. This portability is essential to future use of Triad, in that some distributed applications may require operation in an environment composed of heterogeneous hardware bases.

4 Trade-off Analysis

In addition to the MLS DOS study foundation and the TMach basis, trade-off analysis is a key part of the Triad design approach. Each of the three areas of requirements has ramifications for the other two, and we consider these interactions in terms of three pairs of one area against another, each of which is described in the following sections.

4.1 Real-time Processing versus Security Requirements

In designing the Triad System to provide both real-time processing and satisfy security requirements, several trade-offs must be assessed. There are several areas where security versus real time appear to be in direct conflict, and then there are other areas where the trade-offs required are less dramatic. The following issues in interactions between security and real time are addressed in the Triad design:

Scheduling in any sophisticated scheme provides the potential for covert channels in a trusted system. In some sense, covert channels due to scheduling algorithms are unavoidable. However, there is in fact a very much reduced threat of covert channel exploitation because of certain environmental characteristics of many real-time systems: systems which provide real-time processing are intended for use in a closed processing environment, rather than a general development-oriented OS environment. Such closed environments typically do not support a general programming environment and have extremely limited or no human interaction during real-time processing. These aspects of trusted real-time systems greatly reduce the threat of covert channel exploitation. The Triad approach is to address these issues through a design that satisfies real-time scheduling goals while reducing the opportunity for modulation of covert channels.

Responsiveness versus potential covert channel bandwidth is an issue because real-time systems are designed to be highly responsive to application needs, i.e., provide rapid predictable service times. This is particularly true in the case of the Benefits Accrual (BA) scheduling model which forms the basis of scheduling in the Triad System. On the other hand, this trade-off is potentially challenging because in the secure systems community, the traditional approach to reducing or eliminating covert channels is to add constraints to the scheduler (including introducing non-determinism, partitioning resource usage by security class, and/or using static scheduling). However, the inverse properties (predictability, flexible resource usage, and dynamic scheduling) are all critical to meeting real-time requirements, so the Triad System must be designed to provide a high degree of responsiveness. Therefore, the Triad design identifies potential covert channels, and uses this information to attempt to determine the difficulty of closing or reducing the bandwidth of certain channels.

Flexibility The Triad System design includes a replaceable scheduler interface. The Triad prototype includes a scheduler module using this interface, with appropriate security assurances made for that particular scheduling module, because all scheduling must take place in the Trusted Computing Base (TCB). As a result, any application-supplied scheduler extensions would need to be accompanied by assurance evidence. Specific recommendations will be provided as to what form such assurance evidence should take.

Separation of Mediation from Enforcement TIS research has studied the trade-offs involved in the performance overhead for certain security related operations versus efficient and predictable real-time processing. We have shown that traditional access mediation can be divided into two phases—mediation and enforcement—and this concept plays a central role in the design of TMach security services. Furthermore, we have found that for many real-time systems a considerable amount of access mediation can be done a priori. As a result, mediation can be done as part of the system build process, and enforcement can be done during real-time processing.

Assurance The B3 system architecture calls for a layered, modular TCB of minimal size which uses the techniques of data abstraction and data hiding. The key to satisfaction of this requirement in the Triad system is to minimize changes to the TMach TCB. The modifications Triad will require in the kernel and servers will not affect the existing TMach layering scheme. The addition of the distributed IPC layer will not violate good layering principles, since it will offer services to the TCB layer but not require services of the TCB layer; in turn, the distribution layer will use services offered by the kernel, but not offer any services relied upon by the kernel.

4.1.1 System Build

Central to each of these areas is the use of system-build techniques. Build-time options can be used to control the way that covert channels are addressed, to allow build-specific approaches to providing the kind of scheduling and responsiveness needed for that build, depending on the threats that covert channels pose in the system's actual deployed operational environment. Build-time options can also be used to specify the use of an additional or alternative scheduling module. Also of critical real-time importance is build-time access mediation (described in [4]). The system build process will assist in changing the user-oriented aspects of TMach, e.g. user login and subject creation, to an embedded system orientation which allows security subjects, objects, and their access sets to be pre-defined at system build time.

In addition, the system build process can be used to implement decisions about distribution and replication of some trusted servers. For example, some servers, such as those for Audit and Identification and Authentication (I&A) have databases that need not be replicated in cases where I&A data is static, or where changes are rare and can be handled administratively. In these cases, system build techniques can ensure consistency among copies of the database, without the use of additional run-time functionality.

4.2 Security Requirements versus Distribution

In discussing trade-offs between security requirements and distributed processing requirements, covert channels are, once again, a key issue. Distribution can potentially both exacerbate and alleviate covert channels.

In the Triad System, the most obvious way in which distribution mechanisms introduce covert channels is through the synchronization mechanisms used for coordination of distributed servers on different nodes. These synchronization mechanisms use limited resources that can be exhausted. This can produce a storage-like covert channel in a trusted multi-level server.

Another way in which distribution can exacerbate covert channels is in the area of timing channels. First, accessing resources on a remote node can introduce noticeable delays and therefore introduce possible timing channels. Secondly, in a non-distributed (single node) trusted system, the actual exploitation of timing channels can be difficult because of the noise introduced by many subjects sharing the resources. However, in a distributed system, it is possible that a set of cooperating Trojan Horses on multiple nodes could focus on a covert channel on a single node and exploit that channel more effectively than would be possible on that node alone.

However, just as there is more processing power in a distributed system which can introduce new covert channels, there are also more mitigating factors. First, there are more total resources of the combined system. These resources, if used properly, can make storage or timing channels that result from resource exhaustion harder to exploit because it will be harder to exhaust the resource. In order for this to be true, resources must be managed globally rather than locally or partitioned. Likewise, there is also more noise, from the latency of communications between nodes, which makes more difficult the exploitation of timing channels.

The common feature among these covert channel considerations is that the factors (total resources, resource allocation, communications mechanisms, communication latency) are specific to a particular opera-

tional environment. Therefore, the appropriate measures can only be completely determined at system-build time.

4.3 Real Time Requirements versus Distribution

When considering the trade-offs between real-time requirements and distributed system functionality, difficulty arises because some distributed system operations can take an arbitrary amount of time while real-time operations can have deadlines which must be met.

As a result, the design of the Triad System includes a method of ensuring real-time requirements while preserving distributed system functionality. Specifically, the Triad System provides mechanisms to trade off satisfaction of deadlines versus replication. The Triad System uses a replication protocol based on the Cronus [6] replication protocol. The important characteristic of this protocol which is particularly useful for the Triad system is the ability to vary the amount of computation needed for replication consistency. This approach will allow for considerable flexibility in making trade-offs.

In general, the approach to implementing distribution functionality is to implement it outside the execution paths of potential real-time processing whenever possible (e.g. background processing for consistency updates), and when not possible, to provide features (such as that described above) that allow tuning of the amount of distribution processing that can delay time-bound computation.

5 Triad System Architecture and Features

Triad provides object-oriented services in client-server environment, implemented in a layered architecture. This section describes the architecture and the various real-time and distribution features that are mentioned in the description of the various architectural layers.

5.1 Architecture

The Triad System has a layered architecture consisting of 5 layers:

Kernel Layer: a Mach microkernel, executing in hardware-protected space, and providing the basic abstractions to the rest of the TCB and to untrusted applications. These abstractions include the thread (the schedulable unit of execution), the task (a group of threads sharing an address space and an identity), and message-based IPC (inter-process communication) using protected capabilities called ports. This IPC service is commonly used for local RPC (remote procedure call) between threads of different tasks, and includes the shuttle mechanism.

Distribution Layer: a TCB layer which transparently extends the microkernel's IPC service to operate between hosts, providing the same IPC between threads whether or not their tasks are on the same host. This service is optimized for the cases when it is being used for remote RPC, and includes functionality for scheduling coherence.

System Security Layer: a TCB layer consisting of trusted system servers which provide access to all system resources via access-controlled objects, in accordance with the Triad security policy. Each object has a type, and each type has a specific set of operations (or methods) implemented by a manager for that type. Each manager is a trusted server. As a result, system services are implemented in a client-server, object-oriented manner.

Application TCB Layer: an application TCB layer consisting of trusted servers providing non-system services by managing application-specific types of objects needed by specific applications.

Untrusted Code Layer: an application and OS layer containing untrusted code. Application code may include untrusted clients and type managers, as well as OS servers that implement the services of a specific OS (such as Unix or DOS) on top of TMach's services.

Of these, the first three layers are part of the Triad System; the fourth and fifth layers are application specific. However, a Unix server in the Untrusted Code Layer will be available for application software which uses the Unix application programming interface (API). The affected components of the base TMach system are: the kernel, by the separation of its scheduling aspects into a replaceable scheduling module, and by the addition of the migrating shuttle mechanism for RPC; and two of the servers, where the changes will in no

way affect the TMach's modularity or layering. In addition, a new server will be added to handle distributed IPC (dIPC).

This architecture enables Triad's basic concept of operation, the *invocation*. To use some system or application service, a client invokes a method on an object, resulting in an RPC to a server. The shuttle mechanism ensures the scheduling coherence of the RPC. The distributed IPC service handles cases where the server is located on a different host (or where the server invokes an operation of another server that is a different host) by transparently forwarding the invocation to the other host and handling the scheduling coherence on the other host. The following sections describe this overall functionality in more detail.

5.2 Scheduling

The real-time features of Triad are largely implemented as Kernel Layer functionality of scheduling and thread management. Real-time scheduling is provided by an implementation of Benefits Accrual real-time scheduling algorithm [7]. However, rather than replacing the Mach microkernel's scheduler with a BA scheduler, the Triad approach is to utilize current Mach research in replaceable schedulers [8].

In this approach, the internal structure of the Mach microkernel is altered so that the scheduler implementation is separated from the rest of the microkernel code in a highly modular manner. The resulting interface between the scheduler and the rest of the microkernel is referred to as the replaceable scheduler interface. This interface is general enough to accommodate the requirements of various scheduling algorithms, including BA. Within the scheduling subsystem that implements this interface there is a module that implements a particular scheduling algorithm. This module can be replaced to implement the desired algorithm for a particular system.

The Triad prototype will use a Mach microkernel with a replaceable scheduler interface and replace the standard scheduling algorithm module with a new module that implements the BA algorithm. Thus, even though the scheduling algorithm is different, the rest of the microkernel need not be changed. This is a particularly important result for the Triad System, since different real-time systems have different scheduling algorithm requirements.

Thread attributes are the focus of the thread management aspect of Kernel Layer real-time functionality. Each thread has a number of attributes, some of which pertain to scheduling (e.g. priority in the standard Mach scheduler). As part of the replaceable scheduler interface work, thread scheduling attributes are extended to support a range of scheduling algorithms. In the Triad prototype, which uses the BA algorithm, the thread scheduling attributes which will be used are the those pertinent to BA, including statistical measures of thread activity, as well as hard and soft deadlines.

5.3 Scheduling Coherence

Other than the BA scheduler, the principal Kernel Level real-time functionality in Triad is a mechanism called scheduling coherence. Scheduling coherence is a means to ensure that all the work of a particular real-time computation is scheduled in the same way. This is important when the computation may include an RPC. Consider the common client-server model, where a client makes an RPC, and the RPC is carried out by the server. When a client thread is performing some time-bound computation, the thread's scheduling attributes have been set to particular values so that the computation is likely to complete in time. While a server thread executes the client's RPC, the client thread is sleeping. If the server thread has different scheduling attributes than the client, then the server thread will be scheduled differently than the client thread, perhaps getting a smaller share of processing resources, with the result that the computation may not complete in time.

5.3.1 Shuttles for real-time RPC

We will make use of a recently developed mechanism called shuttle migration [9][10] to provide scheduling coherence for RPC. In this paradigm, the *thread* abstraction is the schedulable entity, comprising the stack and processor state. A *shuttle* comprises the scheduling policy and parameters and resource attributes. During RPC the shuttle migrates to the new task, bringing with it to the new task exactly that information required for scheduling coherence. In the server task a waiting *empty thread* attaches to the migrating shuttle and, now an *active thread*, it executes the appropriate server code to handle the message.

5.3.2 Process Isolation

We are currently analyzing the use of shuttle migration and its effect on TMach's trusted servers. One immediate area of concern is the effect on B3 process isolation and subject definition.

The task is the "process"-like abstraction of the Mach kernel, and the unit of subject definition in TMach, because it is a protected domain of execution—address space, and set of memory object and ports—that is permanently bound to a set security attribute. We need to assure that, though shuttles migrate between tasks, the usual distinction between tasks, and hence between subjects and between isolated processes, is preserved.

When a thread is executing in a task, it has access to everything in the task's address space, and it has some execution context within that address space. When its shuttle migrates to a new task on RPC, that execution state is preserved and left behind, remaining inactive until the shuttle returns to the task. The new task in the RPC chain must have an empty thread for the incoming shuttle to be bound to. When a shuttle enters an empty thread, it carries no state from the previous task other than the thread's scheduling attributes. While the shuttle is executing in the new task, the new active thread has no special access to anything in earlier tasks in the RPC chain. The same is true when a shuttle exits a thread to return to the previous task. In addition, once a shuttle has exited a thread, the exited thread retains no state information.

The security-critical user identity is maintained as well—the user identity associated with a thread is the user identity of the task in which the thread is running. The information flow between the calling task and the called task of a migrating shuttle RPC is restricted to the thread's attributes, the calling task's RPC message, and called task's reply message; and these two messages are exactly the same information that is passed in a traditional RPC.

As a result of all these properties of shuttle migration, all the security-related attributes of the traditional thread model remain with the task-specific thread. Therefore, the critical "process isolation" security requirement is still met, and the definition of the subject is unchanged.

5.4 Distributed IPC

The microkernel provides an IPC service which is the basis of all communication between tasks on the same host. The distributed IPC service is exactly the same service as the microkernel's IPC, but provides for communication between tasks that are on different hosts in a distributed system. The Triad System component that implements this service will be based on the x-kernel implementation of Mach IPC between networked hosts [11].

5.4.1 Distributed IPC Server

The distributed IPC mechanism is implemented by the distributed IPC server. With local IPC, one task sends a message to another using a port held by the other task. The port is a protected capability representing a message queue which is held by one receiving task, and to which messages can be appended by potentially multiple sending tasks. When one task sends a message to another task which is on a different host, the dIPC server is involved as an intermediary. The sending task does the message-send over a port, but the receiver of that port is not really some task on another host; this is not possible because the kernel only knows about tasks on the host it manages. Instead, the receiver of the port is the dIPC server, which receives the message from the sender. This port represents a port on another host, where the message's real destination task runs. The transmission of the message between these two client tasks is the result of the cooperation between the dIPC server on the sender's host and the dIPC server on the receiver's host. In other words, the dIPC server acts as a stand-in for remote tasks that local tasks can communicate with; and local ports received by the dIPC server are stand-ins for ports that those remote tasks are receivers for.

When the dIPC server receives a message over a local port, it checks to see what remote host and remote port correspond to the local port. Then, the message data (and other information including which port it is bound for) is sent over the network to the dIPC server on the receiver's host. This dIPC server sends the message via local IPC on the port received by the actual receiver of the message.

The dIPC server shares critical security-relevant functionality with the Mach microkernel: both are responsible for propagating identification data which is essential for the TMach TCB to enforce access controls. Each TMach task has an attribute called a security identifier (securityID) which represents the user, groups, etc. of the human associated with the task. One important feature of the Mach microkernel's

IPC service is that each message is tagged with the securityID of the sending task. This sender securityID is critical to the enforcement of the security policy by the TMach TCB. The dIPC server transmits the sender securityID with the message data over the network to the remote dIPC server, which sends the original securityID in the message to the receiver of the message. Since the dIPC is separate from the kernel, no kernel modifications will be required, nor will there be an affect on kernel assurance.

5.4.2 Distributed Scheduling Coherence

Another function of the distributed IPC service is implementing scheduling coherence by providing distributed emulation of local thread migration. This emulation is needed because Mach microkernel threads are inherently local abstractions (the Mach microkernel is unaware of other hosts). Hence, a Mach microkernel thread cannot truly migrate to another host. However, the dIPC server can implement a close analog of thread migration.

When the dIPC server is handling an IPC message that is the outgoing part of a migrating thread RPC, some specific actions are needed to maintain scheduling coherence. With each message's data that is transmitted to another host, the dIPC server must also send some more information about the thread that sent the message: its scheduling attributes. This information is used by the receiving dIPC server to perform its distributed emulation of thread migration.

Once a thread migrates into the dIPC server and the outgoing message data is transmitted, the thread goes to sleep. On the receiving machine, the dIPC server uses a local thread to act for the sending thread on the other machine. This thread migrates from the dIPC server to the message's destination task. But before doing so, the dIPC server ensures that this thread has the same scheduling attributes as those that arrived with the message, i.e. the scheduling attributes of the sending thread. As a result, the receiving host's RPC processing is scheduled coherently with the computation in the sending thread on the other host.

5.5 Servers and Replication

Although the basic real-time and distribution functionality is implemented in the kernel and distribution layers of Triad, upper layers also have a role to play.

With respect to real-time, there are four higher-level aspects of real-time functionality. First, for those system servers that can be used by application servers or clients with real-time requirements, the system server implementation must ensure that critical functions are carefully coded so as to predictably execute in bounded time for a time-bound thread. Second, these system servers must use activations, so that time-bound client threads can migrate into the server task to execute server code (see Section 5.3.1). Third, application servers must also use activations when appropriate for the same reasons. Fourth, in order to ensure that each real-time RPC is scheduled coherently, the migrating-thread RPC interface must be used by any application servers and clients which make RPCs and which have real-time requirements.

Thus, higher-level real-time functionality is largely a matter of correctly using lower-level functionality, and this requirement is pervasive for any real-time-relevant code. For distribution, however, the situation is somewhat different. The main lower-level distribution functionality is distributed IPC, and no particular efforts are required of higher-level code to use this transparently implemented service. However, there is more to distributed service than IPC, and it is the TMach system servers that must implement the rest of the distribution functionality.

Of the TMach servers, though, there are only two servers that must provide distributed functionality. These are the Root Name Server (RNS) and the File Server (FS). The RNS is TMach's central security server which implements the reference validation mechanism (RVM) by providing access to all objects and implementing the access controls on them. In addition, the RNS implements some system object types including the directory. The File Server implements another primary system object type, the file. The remainder of the system servers either provide services whose distribution functionality can be handled by system build techniques (see Section 4.1.1), or provide services that are inherently local (e.g. device access) and which have no distributed aspect.

In Triad both the RNS and FS are replicated servers. That is, multiple hosts run an instance of the server, and each server instance cooperates with instances on other hosts, to provide a distributed service. In addition, application-level servers may or may not be replicated, depending on application requirements. If server replication is needed, then the application server implementation can be based on the same server

framework as the system servers, and use the same replication library that will be used to implement Triad extensions to the TMach RNS and FS. As a result of such server replication, the service would be provided by multiple instances of the server. This increases the reliability of the service, and its availability in the face of host failures. Availability and reliability may also be preserved in additional failure modes (such as network partition), depending on the network topology, the amount of server replication, and the degree of cooperation between peer server instances.

5.5.1 Object Replication

The primary distribution functionality of the Triad RNS and FS is object replication. In addition, the RNS uses object replication to provide a global name space. The Triad RNS on each host implements the local TMach name space of that host, and extends this name space into the distributed system, via its cooperation with other RNS instances on other hosts. As a result, the name space is uniform throughout the distributed system, so that every object can be accessed by the same name regardless of which host the accessor resides on.

Object replication is technique used by replicated servers that implement a type, i.e. a set of similar objects and a set of operations on those objects. This technique allows for various degrees of availability of objects, in spite of node or server failures. Without object replication, a server that implements a particular object stores the data of that object, and performs the operations on the object using that object data. The object is globally available throughout the distributed system, since clients anywhere in the system can contact the server for service on the object—subject of course to access controls. However, if that server or its host goes down, then the object is unavailable. Even though the service itself may still be available via other replicated server instances (which manage other objects of the same type), there is no availability of objects that are managed solely by the down server instance.

This sort of single-point-of-failure availability problem can be mitigated by the use of object replication, in which multiple server instances maintain a replica of the object data. As a result, any one of these servers can provide service for the replicated object. Not every object of a type need be replicated, and not every server managing the type has to keep a replica. This, replication can be flexibly used to provide reliable service for an object in potentially several failure modes, depending on the number of replicas, their distribution within the network topology, and the nature of the object's consistency requirements,

The consistency of the object data is the key issue in object replication. By allowing multiple servers to provide service to one object, one allows the possibility that multiple servers could modify their replica's data, causing it to become inconsistent with the replicas of other servers. There are a variety of different consistency mechanisms that can be applied to this situation, but no one of them is suitable for all the different kinds of types of objects that could be replicated—or indeed even for all the objects of one type.

Therefore, the Triad approach is to implement a consistency mechanism that is flexible enough to meet various needs. To do so, Triad has adopted the version-voting mechanism of Cronus and the approach described in [6]. Within this approach, an update to a replica is predicated upon the updating server obtaining locks on other replicas from other servers. The number and/or proportion of all replicas that must be locked is a value that can be specified differently for different replicated objects. Likewise, there are settable parameters for the propagation of new values to replicas that did not participate in an update. As a result of this flexibility, Triad will implement a replication mechanism that is scalable to distributed systems of various sizes. The Triad implementation is in C++, an object-oriented language, in a highly layered and modular fashion. The changes required for replication will be orthogonal to the principal functionality of the servers, and can be accomplished without changing the design or structure of the servers.

6 Conclusion

The Triad project is developing a prototype Triad system designed to provide processing capabilities for real-time distributed military C³I applications which process information of different classifications. Triad uses a base Mach microkernel and TMach trusted servers, for the required multi-level security features. This base is extended with real-time and distribution features which are used to provide services—including migrating thread RPC, distributed IPC, distributed name and file service—which clients and application servers utilize to perform distributed computation meeting real-time requirements.

Our approach is one that minimizes the impact on the assurance and functionality of the existing B3 system base which we are extending. The real-time scheduling microkernel modifications enhance the assurance of the system by increasing modularity and defining security requirements for the use of alternative real-time scheduling algorithms. Because of the layered, modular, object-oriented structure of the TMach servers, the addition of Triad distribution functionality does not impact the existing TMach functionality of the server. The real-time changes to the servers—supporting migrating thread RPC—has very little effect on the existing implementation; the only difference being the initialization code that sets up activations for migrating threads. Indeed, a server can support both styles of RPC—migrating or non-migrating—by setting up both activations and standard threads in such a way that the same RPC operation is executed regardless of which style of RPC the client uses.

Thus, we have started with an existing extensible B3 MLS trusted system base, and established that real-time and distribution extensions can be made in a way that is consistent with the B3 level of assurance. Furthermore, we have identified that there is a small and manageable set of security requirements for the new real-time and distribution functionality—largely restricted to the distributed propagation of security attributes, and the identification of covert channels in real-time scheduling. We have analyzed the tradeoffs between the three areas, and determined the use of system-build techniques to manage these tradeoffs, both in the area of real-time (reducing the threat from scheduling covert channels) and distribution (replicating objects to be local to clients, to avoid incurring network communication overhead in real-time computation). Finally, we have determined that the Triad extensions of TMach are not only consistent with the existing subject/object definitions and policy of TMach, but also supportive of the concepts of operation and programming abstraction identified in the previous study phase of the project.

Therefore, we are confident that we are developing a prototype system that combines trusted MLS services with distributed real-time computation in a manner which supports emerging requirements for modern sophisticated real-time systems.

References

- [1] *Trusted Mach System Architecture*, TIS TMach Edoc-0001-93B, Trusted Information Systems, Inc., 24 May 1993.
- [2] Greenberg, Ira, et al., *The Multilevel Secure Real-Time Distributed Operating System Study*, RL-TR-93-101, Rome Laboratory, May 1993.
- [3] Northcutt, J. Duane, et al, *Decentralized Computing Technology for Fault-Tolerant, Survivable C³I Systems, Functional Description*, 1 December 1988.
- [4] Benzel, T.C. Vickers, et al, *The Role of System Build in Trusted Embedded Systems*, Proceedings of the 13th National Computer Security Conference, Volume I, October 1990.
- [5] Accatta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A., and Young, M., *Mach: A New Kernel Foundation for UNIX*, Proceedings of USENIX, July 1986.
- [6] Floyd, Richard et al., *Future Directions for Replication in Cronus*, BBN Systems and Technologies Corporation, 16 April 1990.
- [7] Jensen, E. Douglas, *A Timeliness Model For Scaleable Real-Time Computer Systems*, Transactions of DECUS, Fall 1992.
- [8] Golub, D., *Adding Real-Time Scheduling to the Mach Kernel*, 1993, unpublished.
- [9] Ford, Bryan, LePreau, Jay, *Evolving Mach 3.0 to a Migrating Thread Model*, Proceedings of USENIX Technical Conference, 17 January 1994.
- [10] Burke, Condict, Mitchell, Reynolds, Watkins, Willcox, *RPC Design for Real-Time Mach*, Open Software Foundation/Research Institute, 12 April 1994.
- [11] Orman, Hilarie, et al., *A Fast and General Implementation of Mach IPC in a Network*, Proceedings of USENIX Mach III Symposium, 19 April 1993.

CONSTRUCTING A HIGH ASSURANCE MAIL GUARD

Richard E. Smith
Secure Computing Corporation
2675 Long Lake Road
Roseville, Minnesota 55113

1. Introduction

Abstract

This paper describes the mail guard constructed as part of the Secure Network Server (SNS) Development Program. The SNS Mail Guard (SMG) provides a highly trustworthy device for transferring electronic mail between networks of differing security levels in accordance with site specific policies. The site defines its message transfer policies based on specific tests of message contents. The development effort pursued high assurance through compliance with trusted software development requirements and through formal assurance of security properties. The resulting mail guard uses the type enforcement capabilities of the LOCK[®] trusted computing base (TCB) to provide the most trustworthy facility achievable with current technology. We have found that high assurance security does not visibly impact mail guard performance.

The Secure Network Server (SNS) Development Program applies the LOCK[®] Trusted Computing Base (TCB) [6] to network security services. The SNS program's goal is to provide a set of useful networking facilities that achieve high security assurance [7]. The first phase of SNS has produced the SNS Mail Guard (SMG), a device capable of controlled reclassification of electronic mail (e-mail). The SMG connects to local networks that use the Internet protocol suite and the Simple Mail Transfer Protocol (SMTP). Users on such networks operating at different security levels can use the SMG to exchange e-mail in a controlled fashion (Figure 1).

Organizations that handle sensitive or classified data generally establish separate computer networks for each sensitivity level of data they must handle. Each network operates in a "System High" mode without security labels to indicate the sensitivity of its data.

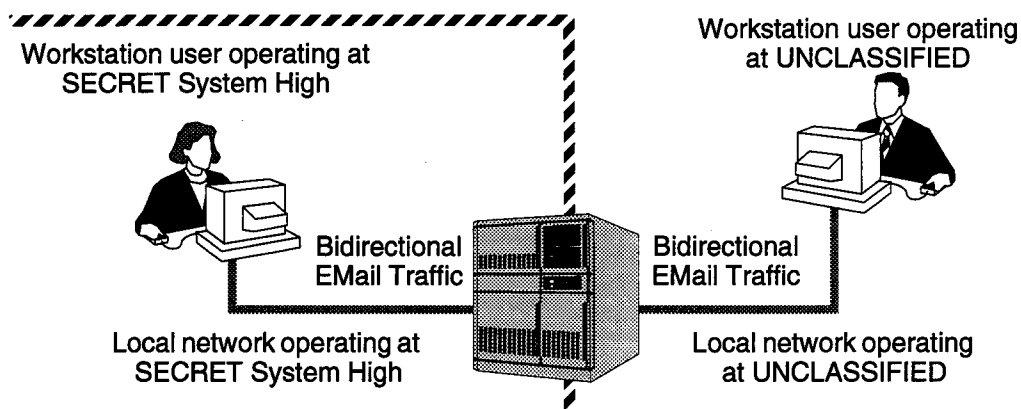


Figure 1: The SMG allows users inside a protected, System High enclave to communicate via Unclassified electronic mail with users outside the enclave.

This is because cost effective, commercially available equipment never provides security labels reliable enough for such applications. By default all data in such networks is implicitly labelled according to the most sensitive data thereon.

E-mail connectivity has become so important that disconnected groups and organizations suffer a recognized operational disadvantage. Today, this is the common fate of groups operating on a classified network. Commercial equipment is not built to keep classified information separate from unclassified. If a classified user composes an unclassified message, there must be a special facility to reliably release the unclassified information to the unclassified network. This facility must be highly trustworthy to prevent the wrong information from flowing between the networks. This is the purpose of the high assurance SMG.

The SMG accepts e-mail messages from one network and, according to the destination address, routes them through a reclassification procedure (Figure 2). If the procedure approves the message for reclassification, then the message is reclassified and passed to the appropriate network for delivery. The decision making process for reclassification is implemented using one or more special procedures called "filters." The choice of filters is controlled according to site specific policy decisions and configured by the SMG's site administration. Different filters may be applied to e-mail traffic depending on the e-mail's source and destination networks.

The architecture of the SMG allows the integration of a variety of filters, depending on the release requirements for the site using the guard. Individuals composing e-mail on personal workstations protected behind an SMG must ensure that the message contents conform to the site's release requirements. Some filters may require that individuals use special software or hardware (like cryptographic services) at their workstations.

The following describes several filters being produced by the SMG program. Filters are individually enabled or disabled according to site security requirements. Each makes its message release decision based on detecting specific types of information in an e-mail message submitted for reclassification:

- SMTP sender or recipient addresses. The filter compares the name of the message's sender and its recipients against a database of addressees. The sender and recipients must all be allowed to send or receive e-mail through the SMG.
- Classification label. The filter searches the body of a message for a line of text indicating the security classification of the message's contents. The author of the message must insert the label into the message to specify the sensitivity of the message's contents.
- Attachment file types. The filter searches the message for attached files in a variety of application specific formats. Each attached file must be of a type that is permitted to traverse the SMG. A site can use this facility to block the accidental importation of executable binary files that may contain virus software.
- Attachment review indicator. The filter searches attached files for a special tag and checksum to indicate that the file had been reviewed by special software (the "attachment review module" or ARM) on the sender's personal workstation. If the site requires attachment review, then the SMG will transmit the message only if attachments it contains have been reviewed using the ARM.
- MOSAIC/MSP digital signature. The filter verifies that the body of the message is formatted according to the Message Security Protocol (MSP) and signed using the MOSAIC digital signature algorithm [3]. The SMG will transmit the message only if the message is signed with a valid signature. The filter may also verify that the signature certificate belongs to an individual authorized to send e-mail through the SMG.
- MOSAIC/MSP encryption. The filter verifies that the body of the message is formatted according to the Message Security Protocol (MSP) and the message text has been encrypted and signed using MOSAIC. The SMG will transmit the message only if its contents are properly encrypted.

If a site decides to allow e-mail to flow in a given direction between two networks, the site must choose which filters will be applied to that e-mail traffic. The choices must maximize the likelihood that reclassification and release decisions are based on information produced by the witting act of an authorized individual rather than on accidental or corrupted contents of an e-mail message. The SMG will typically base its reclassification decisions on information produced or transported by unassured commercial equipment, since that is the equipment in common use today.

The choice of filters, then, depends on the security properties of the networks being connected to the SMG. Small, isolated local networks might use physical security and strong configuration controls to ensure the integrity of e-mail passed to an SMG. Larger, less controlled networks may require the stronger evidence of user identification and message

integrity provided by a cryptographically protected digital signature.

2. Mail guard structure

The SMG combines off the shelf networking software with specially developed guard software, hosting both on the LOCK TCB. A common problem in such systems is to keep the less trustworthy off the shelf software separate from the more trustworthy guard software. It is important to ensure that flaws in the off the shelf software will not prevent the guard software from doing its job. The underlying TCB must protect the integrity of the different software components from one another, and it must ensure that the guard software is never bypassed.

On LOCK, we rely on the Type Enforcement facility to achieve this. Type enforcement is a special form of mandatory, rule based access control provided in

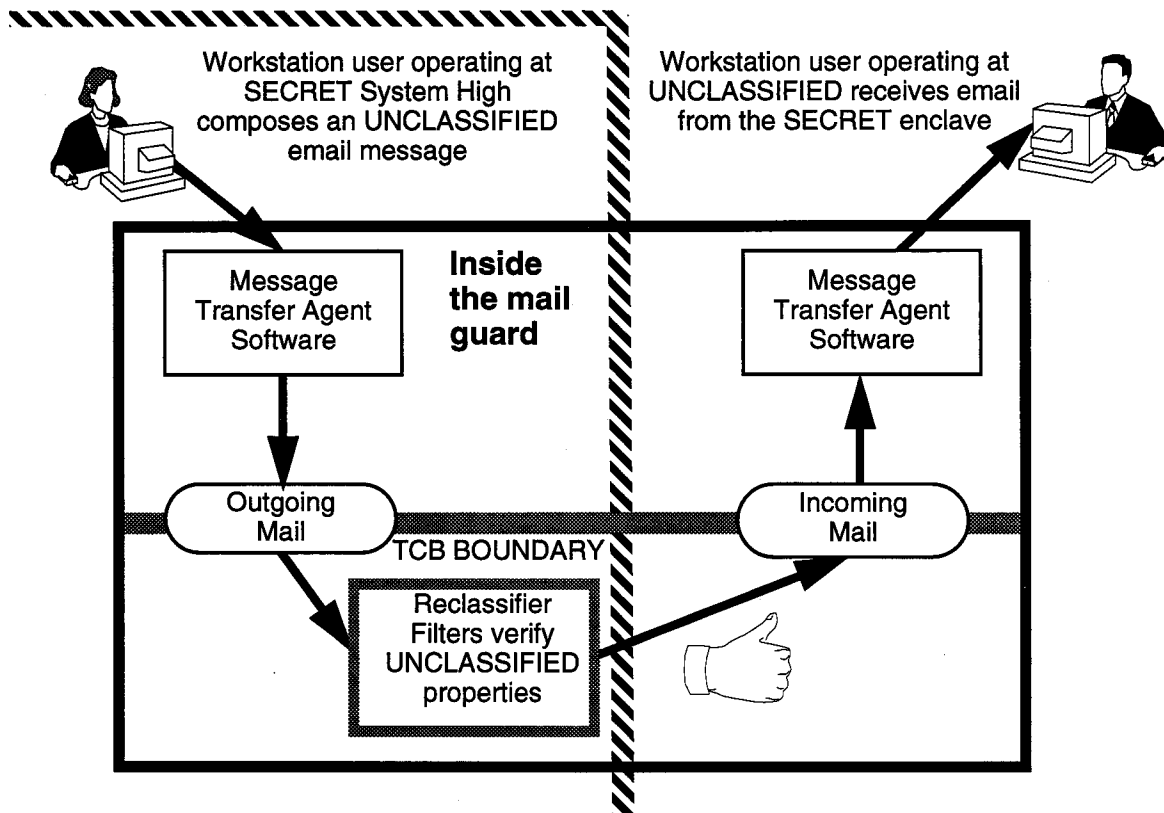


Figure 2: The SMG uses LOCK's type enforcement to isolate the behavior of its off the shelf message transfer agent software from its reclassification procedures.

addition to conventional, label based access control rules. Like access control based on labels, type enforcement completely prevents a program from reading or writing data items unless it is specifically allowed to under the system's security policy. Unlike label based access control, type enforcement is associated with particular programs and particular types of data files.

By placing type restrictions on collections of programs and data items, we can require data to flow through a group of programs in a specific order. This allows us to take data from a program of dubious integrity, pass the data through another program that "censors" it or takes other measures to insure the data's integrity, and then pass it to a third program that assumes the data has the established integrity properties. This technique is called an "assured pipeline" and is fully described in [4].

The SMG implements an assured pipeline to allow its mixture of software components to interact effectively while preserving necessary security properties. Figure 2 illustrates the principal components of the mail guard:

- Message transfer agent (MTA) software
- Reclassifier software (gray bordered box)
- Incoming and outgoing mail queues (ovals)
- Boundary between security levels (dashed line)
- TCB boundary (gray line)

None of the programs illustrated here run in any form of kernel or "root" mode with unlimited access privileges. All these programs are constrained by type enforcement so that they operate as an assured pipeline.

The mail guard pipeline has a message transfer agent at each end and the reclassifier in the middle. The reclassifier itself consists of separate programs in separate domains to enqueue messages for reclassification, invoke the appropriate filters, and to write the approved messages to the incoming queue for the receiving message transfer agent.

The message transfer agents are not allowed to read messages across the boundary between security levels in either direction. This forces all reclassification to go through filters, where virus checks on

incoming executable files and other such activities may occur.

3. Why high assurance?

The purpose of high assurance in the SMG is to ensure that reclassification tests are always performed and never bypassed, regardless of how strong or weak the tests themselves might be. This provides command authorities at the site with certainty that reclassification and release decisions are made in accordance with the specified policy. Furthermore, the authorities must be certain that they can reliably modify the SMG's policy to respond to changes in the network configuration and perceived threat.

High assurance techniques in trusted software development increase the visible trustworthiness of the resulting system. Procedural requirements for trusted software development assure that all developmental steps are carefully thought out and the results are consistently checked. Formal models and analyses of the system's architecture increase the likelihood that all security relevant design flaws have been eliminated. In our experience on the LOCK development program, analysis also uncovers subtle features, design constraints, and interdependencies that would not otherwise have been recognized. This leads to a more thorough understanding of the overall system, its security requirements, and its limitations.

A mail guard is essentially a filter that restricts the flow of data so that "acceptable" messages flow from one security level to another, and "unacceptable" messages do not. Physical security provides assurance that data only flows between the levels via the guard. Walls, floors, protected cabling, and physical access control measures prevent data from flowing between levels except through the guard. The LOCK TCB provides a solid framework which prevents data from flowing between levels except when passing through the filter software. The LOCK formal assurance work provides the detailed look at the trustworthiness and effectiveness of that framework. The mail filters are made modular with respect to the LOCK formal assurance; they are developed and tested separately, and their behavior is analyzed as an extension to the existing system.

It is important to note that formal assurance does not provide some unilateral stamp of quality. All that assurance can do is indicate that the system preserves some specific property. In the case of the SMG, the formal assurance focuses on preservation of label based, mandatory access control on data handled by the SMG.

4. Trusted software development

The SMG development contract mandated that all TCB software be developed in compliance with trust requirements of the Trusted Software Methodology (TSM) developed by the Strategic Defense Initiative Organization [2]. The specific requirements called out in the contract were roughly similar to those associated with Level 3 of the Software Engineering Institute's Capability Maturity Model [5]. The TSM rates software development trustworthiness on a scale from T0 ("untrusted") to T5 ("highly trustworthy"). The SMG program specifies a mixture of requirements ranging from T3 to T5. The T3 requirements primarily apply to environmental and organizational policy issues: properties of the software development environment, for example. The T5 requirements are associated with software development procedures: design analysis, test case development, and formal reviews, for example.

Both contractor and government personnel responsible for the SMG program were trained in the TSM. This training provided the rationale for the requirements and an overview of how they might be applied. The training course made it clear that perfect compliance with the TSM was in fact beyond the state of the art in software engineering. The recommended approach to this problem is to document all requirements that could not be complied with and explain how the program will handle the associated risks.

The SMG program inherited many of its development tools and procedures from the LOCK program, which started in the mid 1980s. This placed practical constraints on the feasibility of incorporating radically new tools or procedures for complying with TSM requirements. An internal review found that it was not feasible to comply with the letter of approximately 8% of the 438 TSM requirements that applied to the program. These requirements were

analyzed with respect to risk to the program, documented, and submitted to the contracting organization for subsequent approval.

An important difference between SMG development philosophy and that of the TSM is the selective application of formal assurance. The TSM implicitly focuses on service assurance: the developed systems must provide faultlessly reliable service and not be sensitive to internal or external denial of service attacks. SMG requirements, on the other hand, focus on the highest possible assurance of security properties. High assurance was not applied to portions of the system whose effects on security could be otherwise constrained. The SMG incorporates "off the shelf" components (hardware and software) and uses a variety of hardware and software mechanisms to constrain their effect on the system's security properties.

The TSM requirements mandated by the contract are applied to all TCB software. This does not include "off the shelf" software, so the TSM was not applied to the TCP/IP software or the message transfer agent (MTA) software. These are constrained by high assurance hardware and software mechanisms so that they can not directly interfere with system security. All software that makes security policy decisions is subjected to formal assurance as well as the TSM requirements.

5. SMG formal assurance

Formal assurance techniques provide the basic evidence that a system is fundamentally trustworthy. It provides a distinctively global view of the system's security properties that can not be duplicated through postdevelopment inspection or test. It also enforces a special discipline on the system development process that focuses early attention on potential security problems. The analytical products of formal assurance provide strong arguments for the soundness of a TCB design, profoundly increasing the TCB's trustworthiness.

SMG assurance focuses on assuring label based access control. The assurance shows that a message will be reclassified if and only if the message presented to the reclassifier is found to be acceptable by whatever filters are applied to it. The assurance

work says nothing about the behavior of entities outside the TCB, which are primarily off the shelf Internet protocol components. Nor does the assurance work guarantee availability or performance. Since the assurance focuses on label based access control, it does not address enforcement of need to know.

The challenge in TCB development is to extend the TCB's capabilities without invalidating previous assurance work. Using LOCK, we proceed by showing that the SMG enhancements either preserve or are constrained by LOCK's existing security mechanisms. The enhancements consist of network device drivers, the TCP/IP protocol stacks, and the MTA software.

Device drivers on LOCK execute in user mode. Each interface is assigned a single security label, so the driver is unable to affect label based access control

decisions. Thus, the network device drivers are constrained by TCB access enforcement mechanisms.

The same approach is applied to the TCP/IP protocol stacks. Separate instances are provided for each distinct security label and each is tasked with processing network traffic at its single security level. The same holds true for the message transfer agent software. Therefore, all off the shelf protocol software is constrained by the TCB and unable to influence label based access control decisions. This limits the amount of analysis required to show that the message transfer agent software maintains label based access control with high assurance.

As discussed previously, authentication and identity based access control for e-mail are not handled by the LOCK TCB. The level of assurance required is subject to site policy depending on the facilities available on the networks served by the SMG.

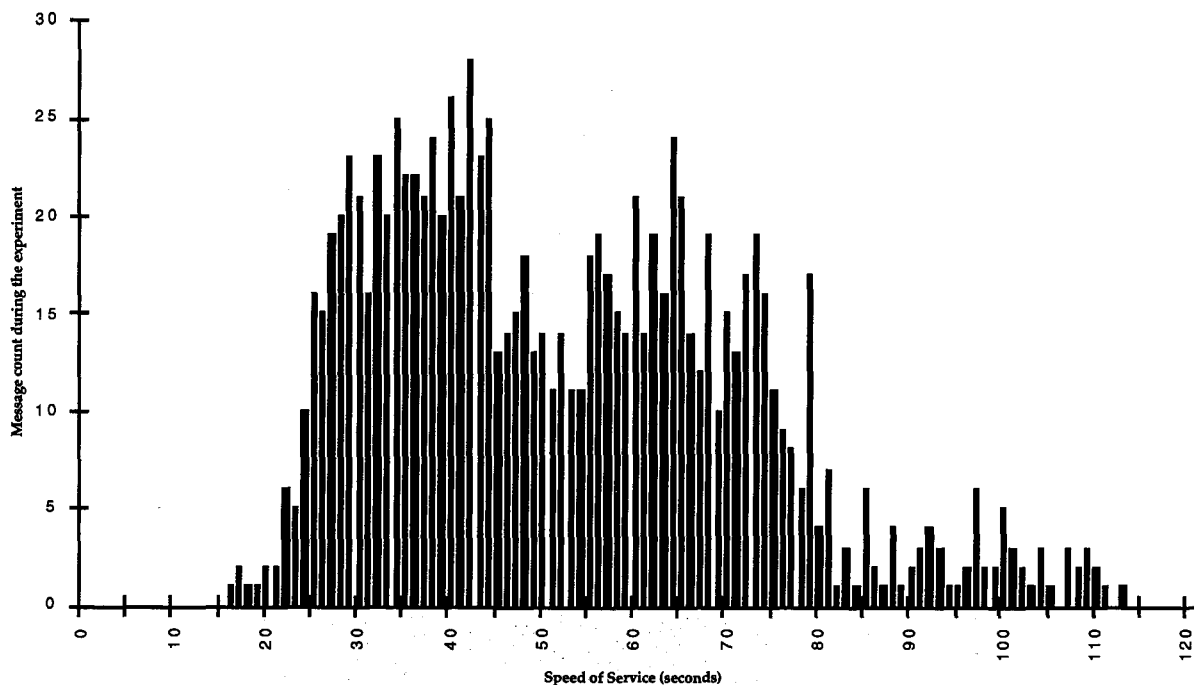


Figure 3: In performance tests, the SMG achieved an average speed of service of 52 seconds (the time to send a message from one host through the guard to another host, with null security filters). Security filtering adds an average worst case overhead of 53%.

6. Performance

An astonishing result of the original LOCK development effort was its level of performance. Standard benchmarks had trouble measuring a consistent difference between the performance of LOCK/ix, LOCK's Unix compatible environment, and that of an unmodified commercial Unix system operating on similar hardware. Uncertainties in benchmarking techniques make all such measurements suspect, but the results make it clear that high security assurance has not significantly reduced LOCK's performance.

Measurements of the SMG exhibit similar performance. Measurements of the MTA software running on the SMG show negligible difference in performance against the same software running on a comparable commercial Unix platform. MTA efficiency is particularly important in the SMG since each reclassified message must pass through two separate MTAs. An independent test and evaluation contractor found that the SMG consistently kept up with commercial mail server software.

Figure 3 shows the results of performance tests on the SMG. These tests measured speed of service while handling bidirectional mail traffic consisting of 10,000 byte messages. The tests did not use security filters. The analysis found the average delivery time ("speed of service") was 52 seconds for messages transmitted across the SMG, with a worst case of 113 seconds.

Filter	Speed of Service (seconds)
No Filtering	52
Access Control Filtering	75
Digital Signature (estimated)	76
Encryption (estimated)	80

The above table compares the SMG's average speed of service when applying different types of security filtering. The first two entries are based on direct measurement of SMG performance. The second two are estimates based on an SMG performance model incorporating the measured performance of the

Tessera crypto card. These figures are all within the speed of service requirements identified for the Defense Message System, providing an order of magnitude or better design margin for incorporating security filters on non-critical message delivery [1].

Acknowledgments

This paper reports the efforts the very capable SMG development team of Secure Computing Corporation's Sever Products Group. The formal assurance work benefited from the valuable contributions of Secure Computing's research organization.

This work was supported by Contract MDA904-93-C-C034. The author also thanks the anonymous reviewers for their constructive comments.

References

- [1] DoD, "Defense Message System (DMS) Required Operational Messaging Characteristics (ROMC)," Department of Defense, 23 April 1993.
- [2] GE Aerospace, "Trusted Software Methodology Volume 1: Trusted Software Program Demonstration, Assessment and Refinement," SDI-S-SD-91-000007, Strategic Defense Initiative Organization, Washington DC, 17 June 1992.
- [3] MOSAIC Program Office, "MOSAIC Program Overview," Version 2, Ft. Meade, MD, 28 January 1994.
- [4] Richard O'Brien and Clyde Rogers, "Developing Applications on LOCK," *Proceedings of the 14th National Computer Security Conference*, Washington, DC, October 1991, page 147.
- [5] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, "Capability Maturity Model for Software," Report CMU/SEI-93-TR-24, Software Engineering Institute, Carnegie-Melon University, Pittsburgh, PA, February 1993.
- [6] O. Sami Saydjari, Joseph M. Beckman, and Jeffrey R. Leaman, "LOCK Trek: Navigating Uncharted Space," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland CA, May 1989, page 167.
- [7] Richard Smith, "MLS File Service for Network Data Sharing," *Proceedings of the Ninth Annual Computer Security Applications Symposium*, Orlando FL, December 1993.

Controlled Execution UNIX*

Lee Badger
Homayoon Tajalli
David Dalva
Daniel Sterne

Trusted Information Systems, Inc.
3060 Washington Rd. (Rt. 97)
Glenwood, MD 21738

Abstract

Although standard mandatory and discretionary access control policies provide control over data, they do not control the identities of programs executed on a given computer system. This aspect of standard access controls is part of the TCSEC [12] security paradigm in which programs are presumed to be hostile and therefore irrelevant to a system's security policy. Controlled Execution (CE) is an access control policy representing a significant departure from this paradigm. A CE policy states the identities of specific programs authorized by the policy to be executed on a given computer system. A CE policy is specified by a trusted system administrator, enforced by a CE Trusted Computing Base (CE-TCB), and mandatory for users and programs.

A suitably configured CE policy confers a variety of security advantages including: 1) binding of processing resources to authorized applications, 2) control over privileged subjects (such as the UNIX root user), 3) heightened penetration resistance and, consequently, 4) enhanced data integrity. CE can be used to strictly control computer virus propagation. A particularly appropriate application of CE is to hosts that connect to the Internet but must not be vulnerable to system or application corruption. Experimentation with a CE implementation based on UNIX has demonstrated that CE can be harmoniously integrated into the UNIX system architecture with moderate effort, broad backward compatibility with existing applications, media, and networks, and enhanced control over the root user.¹

Introduction

Computer systems now manage a vast pool of information assets on behalf of governments, institutions, companies, and individuals. Every month, thousands of computer systems connect to the Internet and to other networks, bringing both increased communications capabilities and increased vulnerabilities. Commercial operating systems (e.g., UNIX variants, DOS, etc.), which have a primary responsibility for preventing intentional or accidental destruction of information assets, typically employ (at best) discretionary access control (DAC) for protection of operating systems and data. In recognition of DAC vulnerabilities to trojan horse attacks [14, 4], in 1985 the U.S. government promulgated a standard, the Trusted Computer System Evaluation Criteria [12], for a class of *trusted* systems implementing mandatory access controls in addition to other security services. Nine years later, however, trusted systems constitute only a tiny fraction of the information infrastructure. Several factors probably have impeded market acceptance of trusted systems, including high cost, the lengthy TCSEC evaluation process, and generally reduced functionality relative to untrusted systems. An additional factor is a perceived mismatch between the security controls provided by trusted systems and the needs of many potential customers. The failure of trusted systems to obtain significant market share motivates an alternate approach. This paper proposes such an approach and presents the

*UNIX is a trademark of UNIX System Laboratories, Inc.

¹This work is sponsored by ARPA/CSTO under ARPA order #7311, contract MDA972-90-C-0027.

concepts, costs, and benefits of a security policy and implementation strategy providing practical security that is compatible, effective, and affordable.

Controlled Execution² (CE) is an access control mechanism that constrains a system to execute *only* the application programs that are members of a set of *authorized* application programs specified by a Trusted System Administrator (TSA). A fundamental and highly assured portion of a CE system is responsible for carrying out TSA's commands. A CE system enforces the specified CE policy on all other users and on all programs. If the behaviors of all of the installed programs are understood, the behavior of a CE system can also be understood: the CE system's behavior is a composition of the behaviors of its installed programs. In this aspect, CE represents a significant departure from the TCSEC MAC paradigm in which application programs are presumed to be malicious, nothing about their behavior is assumed, and the purpose of a TCB is to impose a restraining policy on the data accesses that the "unknown" application programs may attempt. In contrast, CE explicitly addresses what we believe to be a current security reality: *application program behaviors are security relevant*. This reality surfaces continually in day-to-day operations. For example, a simple security policy that a purchasing officer is only authorized to approve checks up to a maximum amount can be enforced at the operating system level *only* by restraining the set of applications to those enforcing the required constraints. A suitably configured CE policy confers a variety of security advantages including:

computing resource control Specific programs can be bound via CE to specific computer systems (CPUs', memory, devices, etc.). These hardware and software resources are bound to the installed program set and cannot be diverted for unauthorized applications.

control over privileged subjects Privileged subjects (such as the UNIX root uid) are confined to execute *only* the installed applications. This reduces the possibility of a trojan horse or virus penetrating a system because the malicious code would have had to be installed as an authorized application. Additionally, CE prevents even privileged subjects from corrupting the system or its approved applications either intentionally or accidentally.

heightened penetration resistance A significant class of system penetrations depends on the attacker's ability to install and execute malicious programs on the target system. The ability to run malicious programs exposes the system call interface to attack, possibly exploiting low level security flaws in the system. This class of penetrations is closed by CE.

enhanced data integrity All data is processed by known, authorized programs. This control over programs executed enhances data integrity since unauthorized and perhaps malicious programs never have an opportunity to access data. One example of a malicious program is a program that has been infected by a computer virus. A virus can only execute on a CE system if the (already) infected program has been installed by the TSA as an authorized application. Even if a virus is accidentally installed on a CE system, CE prevents further propagation because the virus is not able to alter other installed applications.

The CE policy of limiting execution to programs in a preauthorized set may appear to be overly restrictive; however, our experience with a UNIX prototype has indicated that CE has only minor usability impact for many activities. Although not appropriate for every system, CE is applicable to many systems that have relatively stable application sets, including document processing and email systems, file servers, and many compile-only software development machines. CE is appropriate in many contexts where users have no need/authority to install or change programs, for example: turnkey systems used by banks and hospitals, computer controlled manufacturing, financial accounting, etc. A particularly appropriate application of CE is to hosts that connect to the Internet but must not be vulnerable to system or application corruption. Our prototype experience indicates that CE can be harmoniously integrated into the UNIX system architecture with moderate effort, broad backward compatibility with existing applications media and networks, and

²Patent Pending.

enhanced control over the root user. In the context of the current dichotomy between inexpensive, DAC-only commercial operating systems and expensive trusted operating systems providing MAC, CE appears to offer the framework for a much needed middle ground of security-enhanced yet cost-effective systems.

The purpose of this paper is to present both the advantages and disadvantages of the CE policy and a specific application to UNIX. We first explore CE concepts and then interpret those concepts for the UNIX system architecture. Next we discuss the design and implementation of the CE UNIX prototype. Finally we review related work and preview future directions.

Controlled Execution Systems

A CE system enforces an access control policy over two kinds of system events: program installation and program execution. A CE system enforces its access controls regardless of the actions taken by an (erroneously) installed application. To provide this policy, a CE system must be able to protect its own integrity as well as the integrity of the authorized applications. Additionally, a CE system must have control over the API that controls program execution (e.g., the `exec()` family of system calls in UNIX). Figure 1 shows the general structure of a generic CE system. A CE system manages hardware devices and memory, providing (and controlling) a system call interface. Although figure 1 may suggest a monolithic operating system kernel, CE can be applied equally well to micro-kernel based operating systems such as OSF/1 MK. A CE system must have available to it a set of locally controlled devices that together constitute the Protected Media (PM). The PM is usually implemented by a disk device but may be any persistent storage device. The PM stores the CE operating system image, any internal persistent space needed by a CE system as it runs (e.g., swap space), and all the installed applications. The special characteristic of the PM is that it can *only* be modified by the TSA and by the CE operating system for internal bookkeeping. This access control applies regardless of any privileges possessed by application processes of the host operating system. We designate the portion of a CE system that is responsible for enforcing the CE policy the CE Trusted Computing Base (CE-TCB). The CE-TCB includes the PM, at least one console device, the operating system image that resides on the PM, and a small set of utility programs for installing authorized applications and performing maintenance operations on the PM.

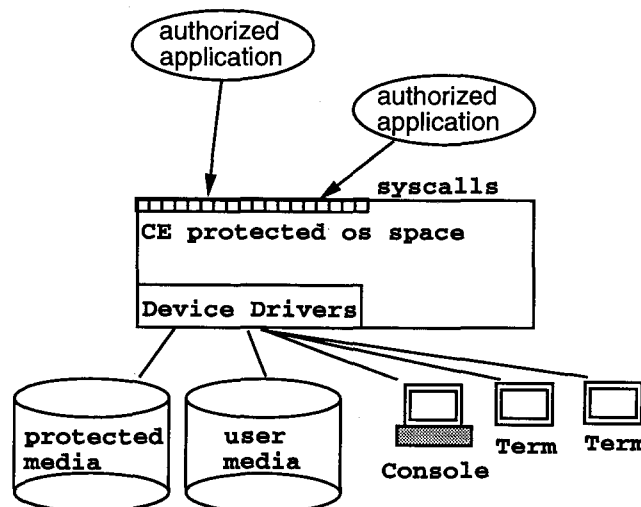


Figure 1: General CE Structure

To allow a TSA the ability to install applications, a CE-TCB must be able to reliably authenticate the TSA and ensure that other users are not able to impersonate TSA requests. A variety of strategies exist

for establishing this trustworthy connection between the TSA and a CE system, most notably the Trusted Path mechanism discussed in the TCSEC [12]. A trusted path for CE may be significantly simpler than that required for the TCSEC. For the TCSEC, the trusted path must identify all users to provide accurate auditing and to ensure that users do not penetrate other user accounts. This requires a trusted path that is available for all users. In contrast, the CE trusted path need only distinguish between the TSA and all other users. Depending on the level of convenience required for the installation of new programs, the CE trusted path can be implemented with different combinations of system support and administrative controls.

Once a set of authorized programs has been installed on the PM, authorized programs and users may execute them. Any attempt to execute a program that does not reside on the PM results in a CE access control denial. In the context of CE, program execution is a program file activation event (e.g., for UNIX, the `exec()` system call) followed by a sequence of program events. For a CE-TCB that exports a particular application programming interface (API), both the activation and the successive events are calls to the API. Depending on its input, each program is capable of a set of *behaviors* where a behavior is a particular sequence of program events. A CE access control policy bounds the set of behaviors that a system permits; a CE-TCB behavior is a composition of the program behaviors for the system's authorized programs.

Interpreters and Trust in Applications

An inherent limitation of CE is that a CE-TCB can only enforce execution control *at its exported API*. If an authorized application contains an interpreter (e.g., a shell), programs interpreted on the interpreter appear as data to the CE-TCB and will not be controlled. This problem primarily affects the CE benefits of controlled computing resources and enhanced data integrity: an interpreter could use processing power for unauthorized applications and possibly corrupt important data. This is potentially a significant problem because many widely used programs contain interpreters with various levels of programmability. This problem can be addressed at various levels of cost and assurance by authorizing only programs that: 1) do not contain interpreters, 2) contain restricted interpreters, or 3) interpret only files stored on the PM. This problem does not significantly reduce the CE benefits of control over privileged subjects and heightened penetration resistance because the CE-TCB integrity and the integrity of approved applications cannot be affected by interpreters.

A related, important issue is that a TSA must decide whether or not to install a program based on its anticipated behavior. If a single malicious program is installed, it could behave in an arbitrary manner and therefore could potentially divert computing resources to unauthorized uses or damage data. However, as in the case with interpreters, a malicious installed program cannot alter either the CE system or other installed programs. The primary defense is to obtain programs only from trustworthy providers. A number of strategies can be used to increase trust in the programs, including testing and source code analysis. While absolute knowledge of a program's behaviors may not be possible using those means, a software selection can be based (as it commonly is) on a probability, or reasonable expectation.

Controlled Execution UNIX

A CE UNIX (CEU) system is a UNIX system that has been modified to enforce (perhaps in addition to TCSEC policies) the CE policy. The purpose of this section is to present a design and rationale for a CEU system. There are many variations of UNIX representing deviations and extensions to the "traditional" UNIX concepts and structures. The design presented here is based on the features and structures of "traditional" UNIX (hereafter, just UNIX). This section will first briefly review UNIX structures, identify UNIX components that must be inside the CE-TCB, discuss required modifications to UNIX components, and finally discuss resulting implementation assurances.

Figure 2 shows the general UNIX system structure. The UNIX kernel manages physical memory, one or more CPUs, and a set of connected devices. The kernel provides the process abstraction by timesharing its CPUs and implements file and file-system abstractions on connected devices. UNIX processes (in figure 2, `init`, `P1`, and `P2`) manage the only threads of control. A UNIX process is generally a file that has been

set in execution by the UNIX kernel. Every file is either a device-special or a regular file.³ In either case, the kernel implements a file-oriented interface that allows (and confines) processes to interact (in figure 2, dotted lines) with devices in a structured manner. Processes interact with the kernel, each other, and all devices indirectly through system calls that are mediated by the kernel. Generally, an attached device can be treated as a file; the kernel provides system calls to open, read, and write these files. Additionally, some devices are interpreted by the kernel as file systems (in figure 2, Root FS, FS 2, and FS 3). A file system is a collection of files that are related via a hierarchical naming scheme; file systems allow processes to view portions of devices as discrete data containers.

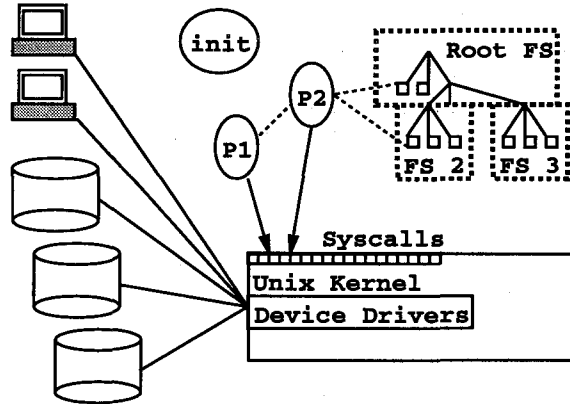


Figure 2: General UNIX Structure

Informally, a UNIX CE-TCB can enforce execution control by requiring that all programs that can run reside on special devices over which the kernel has complete control. The collection of such devices is the protected media (PM). The system can ensure that files are only added to or removed from the PM by the system administrator and that only the system administrator can modify files on the PM. In addition to providing files and file systems to the rest of the system, the system uses the PM for its internal purposes (process swapping and paging); this prevents implementation dependencies on untrustworthy media. The components of the example UNIX CE-TCB are:

PM At least one local secondary storage device under local control of the CEU operating system.

UNIX kernel The CEU image stored on the PM.

Init process This process is the master coordinator for other processes. This process brings the system to multiuser or single user mode.

cp program When used from the trusted path, copies files.

rm program When used from the trusted path, removes files.

command shell When used from the trusted path, processes interactive user commands.

system programs A small set of programs required for maintenance of the PM (primarily, the fsck program).

Other UNIX components may be included for ease of use, but the policy only *depends* on these.

³Note we are ignoring special files such as named pipes, directories, FIFOs, sockets, and symbolic links because their properties are not relevant to general discussion of CEU.

Because every request to execute a new program is made to the CE-TCB and because the CE-TCB controls the contents of the PM, the CE-TCB is able to enforce the execution control policy on those requests. The kernel creates and manages all processes and can therefore mediate process create requests. The kernel's mediation ensures that processes only execute if their executables are stored on the PM. The kernel is able to perform this mediation because it has access to the (device) locations for executable files at process execution time.

In order to enforce the CE policy, the CE-TCB must protect itself from modification. Because the UNIX system provides address translation, processes cannot directly modify the kernel. By ensuring that the kernel only depends on the PM for its correct operation and by protecting the PM from modification, the CE-TCB can protect itself. Processes can access devices (including those that constitute the PM) in *only three different ways*: 1) accessing the device special files that represent physical devices, 2) file-oriented operations on files contained in file systems that represent attached devices, and 3) changes to the kernel's internal memory management (swapping) policy using special system calls to alter the configuration. ⁴ The CE-TCB protects itself from all three possibilities through small changes in the kernel calls that manipulate device special files and file systems. CE access control falls into two general categories: 1) if a UNIX object exists on the PM, prohibit modify access, and 2) if a UNIX object does *not* exist on the PM, prohibit execute access.

A number of mediation points are required in the UNIX API to implement the CE policy. A high level list is:

exec, execv, execve These functions are only enabled when the program to be executed resides on the PM. These functions preserve the CE policy because the PM only holds system programs and approved applications.

creat, mknod, mkdir, rmdir, unlink, rename, truncate, chown, chmod These functions are only enabled when the object to be created or modified does not reside on a read-only file system of the PM or when the function is issued by the TSA.

open This function is only enabled when the flags argument to open() does not include "write," when the object to be opened does not reside on the PM, when the system is initializing, or when the function is issued by the TSA. This function helps to preserve system integrity by guarding all other functions that operate using file descriptors returned by open.

swapon This function instructs the kernel to use the passed devices for swapping. This function is only enabled for devices that constitute the PM; swapon can therefore not be used to compromise the system's integrity or to violate the CE policy.

write This function is only enabled following an open for a file that specifies write access. The open and creat calls guard this operation so that the CE policy is preserved.

ioctl This function implements many operations on devices. The operations are categorized into reading and writing operations; an open or creat call must have returned a descriptor open for writing for the writing operations to be used. This function is therefore guarded by the open and creat functions.

ptrace This function is disabled because it would allow one application to insert arbitrary code into another.

A primary UNIX benefit from CEU is reduction of the strength of the powerful UNIX root privilege while maintaining most of its usefulness. The root privilege is currently a significant UNIX security weakness. Many applications need to run as root; the presence of many programs holding the root privilege results in a weakest-link phenomenon in which any root-privileged program can, through mistake or intent, compromise the integrity and security of an entire system. A number of systems and standards [11, 9, 13] have separated

⁴Newer versions of UNIX may implement additional means for accessing devices, such as the provision for dynamically loadable device drivers; such additional mechanisms would also have to be controlled by a CE-TCB.

the root privilege into a set of privileges to more closely implement the principle of least privilege. This approach is not ideal, however, because the root privilege still exists as a collection of separate privileges and because one privilege can sometimes be used to obtain others.

CEU makes a substantial departure from the typical strategy for dealing with root. Because the CE policy is compatible with standard UNIX applications, there is no need to provide privileges to suspend the policy. This allows the CE policy to be enforced even on the root user, making the root privilege irrelevant for the CE policy and the CE-TCB. The CE UNIX interpretation therefore protects against misuse of the root privilege in two ways: 1) only authorized applications can execute, and therefore the use of the root privilege is confined to them, and 2) even if a malicious application is installed, the CE-TCB protects itself from any modifications, regardless of privilege.

UNIX Prototype

In order to validate the controlled execution concept and its interpretation to UNIX, a prototype-CEU system was developed. The prototype is based on the OSF/1 MK operating system running on an HP Apollo 9000 model 710 PA-RISC Desktop Workstation. The implementation effort required was approximately 5 staff weeks.

Two sets of changes to OSF/1 implement the controlled execution prototype. The first set of changes controls modification of the protected media. The prototype implements a protected media by checking write access to a designated set of disk partitions and other critical device special files at the system call interface. The identities of these devices are compiled into the kernel as a table and consulted by an internal mediation function as processes access system resources through the API. Access decisions are based on the state of a variable that denotes the *run state* of the system. Three *run states* are defined: booting, single user mode, and multiuser mode. *Run state* support includes a new system call to set the *run state* of the system, with the limitation that it can only increase (e.g., single-user mode to multiuser mode). When the system is booting, write access is permitted to the protected media but only through communication with a cryptographically authenticated control facility. The boot state therefore implements a trusted path to the TSA. After the system has completed the boot sequence and goes into multiuser mode, all attempts to modify the protected media are denied. Single user mode is treated the same as multiuser mode in the prototype.

The second set of changes limits execute access to programs residing on the protected media when the system is in single or multiuser mode. Very few system programs were changed:

Init The *init* program is the first process. Since it controls the booting of the system, it changes the *run state* when the system enters a new mode. It was also modified to mount the writeable */root* file system (described below) immediately in conjunction with mounting the protected media root filesystem.

Passwd The *passwd* program was modified to use the */root* filesystem, where the *passwd* file now resides (ie., */root/etc/passwd*).

Vipw Like the *passwd* program, *vipw* was modified to use the */root* filesystem for temporary storage.

Because the protected media is read-only after the system has booted, the file system organization had to be altered both to permit other system configuration (e.g., changing the system startup script in */etc/rc*, adding users, backup) to be performed without writing to the PM and also to maintain compatibility with programs that expect to find and modify specific files and directories on UNIX systems (e.g., */tmp*). A new file system, */root*, was introduced to hold these writeable entities and to represent the normal UNIX root file system. During system bootup, the *init* program mounts */root* read/write so that other programs can find it. The standard */usr*, */tmp*, */etc*, */dev*, etc. directories were created under */root*. Symbolic links were then created in the protected media that point to these directories on the read/write root. This allowed some links to be replaced by writeable files on the read/write root, therefore allowing old programs to find writeable versions of files in the normal UNIX locations.

With the exception of several system programs that were not converted, the prototype behaved like an unmodified UNIX system. The standard programs ran normally, including the compiler, but newly generated executables, or copies of executables from the protected media, could not execute.

Penetration Testing Experience

To validate the soundness of the prototype, a 40-hour penetration study was conducted by an experienced UNIX programmer with complete access to the prototype source and consulting from the prototype's designer. The penetration effort was conducted with two sets of rules: in the first, the attacker could use only installed applications. In the second, the attacker had use of a subverted application that allowed the attacker to stress the system's system call interface. In the second case, the attacker attempted to gain control over the entire system. Under the first assumption, the attacker discovered two weaknesses that allowed unauthorized modifications of the PM:

- The last-modified inode fields for files on the PM could be updated using the touch command (which uses the utimes() system call). This trivial modification of the PM resulted from incomplete mediation in the prototype.
- The Network File System (NFS) subsystem in the OSF/1 MK system uses its own access control functions. These functions had not been modified in the prototype to prevent modifications of the PM. This more significant vulnerability also resulted from incomplete mediation in the prototype.

Under the second assumption, the attacker discovered that the OSF/1 specific facilities for loadable kernel modules and calling functions in loaded modules could be misused to take control of the OSF/1 system. These penetrations also resulted from incomplete mediation.

All of the penetrations were easily correctable and resulted from the prototype status of the tested CEU system. None of the penetrations indicated conceptual problems with CEU.

Related Work

Systems that restrict execution comprise the most closely related work. Many systems allow some restrictions to be placed on which programs can be executed, however the authors know of no existing system that makes execution control fundamental in that it is administered and enforced independently of other security mechanisms and (most) system administration activities. Several access control mechanisms are relevant to CE. CE could be implemented using Domains and Types [3] by marking all executable programs with a type *T* and ensuring that no domain holds execute access for any other type and also that no domain holds modify access for *T*. The integrity model proposed by Clark and Wilson could also be used to express a CE policy. In that model, a subset of a system's data objects are labeled as constrained data items (CDIs); the system maintains a set of authorized programs that are permitted to operate on the CDIs, the transformation procedures (TPs), and the users that are permitted to run the TPs. Very few systems have implemented either Domains and Types or mechanisms corresponding directly to the Clark/Wilson model. Complexity of administration and implementation expense may account for this. Although these access control technologies have clear value, when only execution control is required, we believe that a product-quality CE implementation can offer similar execution control at lower cost.

Virus countermeasures are also relevant to this work. As summarized in [5], virus countermeasures fall into four groups: prevention, detection, containment, and recovery. Virus detection can be performed by scanning executable files for particular virus signatures, by computing checksums on executable files, or by comparing program behavior to expected behavior as determined by an analysis of program source code. The major deficiency of these forms of detection is that virus code may execute before a virus is detected. When signature scanning is used, viruses whose signatures or behaviors are not known to the detection program may not be detected. Checksums may be vulnerable to viral attack. Similarly, errors in analyzing program behavior may permit a virus to spread undetected. Once detected, containment is the process of limiting the

further spread until the virus can be removed (recovery). Containment and recovery often require draconian measures such as isolating infected systems until "clean" versions of programs and data can be restored from backup media. As this can be expensive and time consuming, prevention is the preferable strategy where it is feasible. In the absence of a CE environment, virus prevention is mainly attempted through risk-reducing administrative controls that isolate systems from potentially malicious programs or keep track of suspicious network patterns [5]. CE-TCBs provide stronger control over the introduction and containment of computer viruses and worms because only system administrators can install or modify executable programs. This allows CE-TCBs to remain connected to open networks while preventing infection due to accidental or intentional installation of a virus by non-administrative users. CE provides a more useful prevention technique because systems need not be isolated to prevent infections. As an example, the internet worm [8] would not have spread to CE-TCBs.

Future Directions

Two new ideas have emerged from the CEU work. The first is the use of CE to bind software to hardware so that hardware suppliers can target (and price) hardware for specific customers and applications. The second is that, along with tamper-detectable or tamper-resistant packaging, CE may support remote monitoring facilities and allow the export of some high performance systems that could not otherwise be exported. Currently, some supercomputers must be constantly manned to ensure that they have not been diverted to unauthorized applications. CE may provide the same assurances without need for expensive and constant human monitoring. Research is ongoing [15] on the formulation of both software and hardware protection measures and the corresponding organizational support (to ensure conformance on the customer's part) to provide this capability at reasonable cost.

Other applications include combining CEU with MAC and improved accountability to reduce opportunity for exploitation of covert channels.

Conclusions

CE appears to be an extremely powerful policy for binding specific computer systems to specific authorized applications and for ameliorating system and data integrity threats. CE provides both a means to prevent unauthorized programs from running and a guarantee that, even if an authorized program contains malicious or erroneous code, the program cannot damage the CE-TCB. This provides a stable system on which to implement other policies such as disclosure and integrity policies. A particular benefit of CE-TCBs is that they are immune to typical virus infections. The internet worm [8], for example, would not have infected CE-TCBs because the worm's spreading technique depended on the ability to compile and execute malicious code on target machines.⁵

The functional requirements of Controlled Execution indicate that the policy may have wide applicability to operating systems. The UNIX interpretation of CE is compatible with standard UNIX concepts, requires few source code changes, and provides increased system integrity even in the presence of (installed) malicious root programs. This effectively addresses a major UNIX security weakness. The combination of increased control, high compatibility, and low cost appears to offer a middle ground for operating systems that is less costly than trusted systems and more resistant to a variety of threats than current untrusted systems.

*

References

⁵However, the worm might have compromised user accounts on CE-TCBs by guessing passwords. Such authentication attacks are not preventable using only access controls.

- [1] D.E. Bell and L. Lapadula, *Secure Computer System: Unified Exposition and Multics Interpretation*. (Technical Report No. ESD-TR-75-306, Electronics Systems Division, AFSC, Hanscom AF Base, Bedford MA, 1976).
- [2] K.J. Biba, Integrity Considerations for Secure Computer Systems, USAF Electronic Systems Division, Bedford, Mass., ESD-TR-76-372, 1977.
- [3] W.E. Boebert and R.Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies," Proceedings of the 8th National Computer Security Conference, Gaithersburg, Md., P. 18, 1985.
- [4] W.E. Boebert, and C.T. Ferguson, "A Partial Solution to the Discretionary Trojan Horse Problem," 9th Security Conference, DoD/NBS, September 1985, pp 141-144.
- [5] D.M. Chess, "Computer Viruses and Related Threats to Computer and Network Security," Computer Networks and ISDN Systems, Vol 17, 1989.
- [6] D.D Clark and D.R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland, Cal., 1987.
- [7] F. Cohen, "Computer Viruses," Proceedings of the 7th DoD/NBS Computer Security Conference, 1984.
- [8] P.J. Denning, "The Internet Worm," *American Scientist*, Vol 77 March-April 1989.
- [9] Final Evaluation Report of American Telephone and Telegraph System V/MLS Release 1.1.2 Running on Unix System V Release 3.1.1, Oct. 18, 1989, CSC-EPL-89/003.
- [10] J.A. Goguen and J. Meseguer, "Unwinding and Inference Control." Proceedings of the 1984 IEEE Symposium on Security and Privacy, 1984.
- [11] G.L. Luckenbaugh, V.D. Gligor, L.J. Dotterer, C.S.Chandersekaran, N. Vasudevan, "Interpretation of the Bell and Lapadula Model for Secure Xenix," Proceedings of the 9th National Computer Security Conference, Sept. 1986, p113.
- [12] National Computer Security Center, "Department of Defense Trusted Computer System Evaluation Criteria," DoD 5200.28-STD, December 1985.
- [13] POSIX 1003.6 Draft 12, Sept. 1991.
- [14] M.D. Schroeder, "Cooperation of Mutually Suspicious Subsystems," PhD dissertation, M.I.T., 1972.
- [15] D. Sterne, "Safeguards and Supercomputers," TIS Technical Report TISR 461D, Aug 24, 1993.

ARCHITECTURES FOR C2 DOS/WINDOWS-BASED PERSONAL COMPUTERS¹

Securing an "Unsecurable" Operating System

Jeremy Epstein, Gary Grossman, Frederick Maxwell,
Noble Veirs III, Albert Donaldson, Cornelius Haley²

Cordant, Inc.
11400 Commerce Park Drive
Reston Virginia 22091

ABSTRACT

DOS and Windows-based personal computers have many well-known security problems. While there are many products which offer security features for DOS/Windows, assurance is notably lacking. A variety of solutions to meet the Trusted Computer System Evaluation Criteria (TCSEC) Class C2 requirements for both functionality *and* assurance are presented along with descriptions of advantages and disadvantages of each solution. While few of the approaches described here are new, we show that architectures which are theoretically suitable are not practical, and perhaps the "best" solution is obtained through careful balancing of tradeoffs.

1. Introduction

Personal Computers (PCs) based on MS-DOS [1,2] and related systems³ are known to be

¹Copyright © 1994 Cordant Inc.

²Email addresses for the authors are: Epstein: jepstein@cordant.com; Grossman: ggross@cordant.com; Veirs: noblev@pulse-sys.com; Donaldson: al@escom.com; Haley: nhaley@cordant.com.

³Henceforth, we will refer to MS-DOS and related systems such as Novell DOS collectively as DOS systems, specifying the variant when relevant. We include Microsoft Windows, which acts as an extension to DOS.

highly insecure and cannot be secured without major architectural changes, primarily because they do not provide any separation between the operating system and applications. Because they are so widely used, it is desirable to extend the PC hardware and software architecture to meet the Trusted Computer System Evaluation Criteria (TCSEC [4]) Class C2 requirements. There are many ways to accomplish this task, depending on the resources available for developing the solution and the acceptable impact on the end user.

This paper describes the DOS/Windows architecture and explains its fundamental problems. It then summarizes the requirements for a C2 system and describes a variety of approaches to meet the C2 criteria.

2. DOS Architecture

DOS is a highly extensible operating system for personal computers. It is routinely extended to support varying types of I/O devices and new applications. While it is intimately tied to the Intel 80x86 family of processors, it is an "open" system in that specifications for extending it are public⁴, and many companies build extensions.

⁴In many areas the "openness" of DOS is not by design, but rather because many ingenious developers have made the effort to figure out parts of the interface and to publish their findings. This has become more problematic as new interfaces become increasingly complex.

In a DOS system, the operating system and all applications share a single address space. Applications can request services from the operating system, from the Basic Input Output System (BIOS), or directly from the hardware (typically by executing IN and OUT instructions to read and write device registers). While memory management is not built into DOS, it has been added using conventions such as the DOS Protected Mode Interface (DPMI [3]) specification, which describes how cooperating applications share the processor and memory. Unfortunately, use of DPMI services is simply a convention, and is not followed by many applications. Many popular applications, including Microsoft Windows, do not adhere to the DPMI conventions⁵.

Newer members of the Intel 80x86 family (e.g., 80386, 80486, Pentium) include a facility to create virtual 8086s. However, there is no capability to create virtual 80386⁶ systems. In particular, unprivileged instructions can be used to detect whether the software is running in the most privileged level of the hardware. As a result, software can detect whether it has complete control of the machine and refuse to operate if it does not. Additionally, memory management hardware which could be used to provide virtual 8086 boxes is used by applications such as Windows for their own purposes. Hence, any attempt to create virtual systems is likely to encounter compatibility problems with existing applications.

From a security perspective, DOS is a single user system which has none of the basic security features common in larger systems. All security is based on physical access to the computer. There is no concept of system logins or of access controls. By comparison, even

⁵Applications which use DPMI services can run under Windows, but Windows itself cannot run under a system which provides DPMI services.

⁶We will use the term 80386 to refer to the 80386, 80486, Pentium, and compatible processors.

commercial versions of UNIX and VMS include identification and authentication facilities, access controls, and have operating system kernels which can meet the reference validation requirements for trusted systems. Thus, building a trusted DOS system is a much larger hurdle than building a trusted UNIX or VMS system, even at moderate levels of assurance.

In DOS, the operating system can be manipulated by applications without limitation. Because applications share an address space with the operating system, applications frequently modify operating system data structures to take control of interrupts. For performance reasons, applications also bypass the operating system and make direct accesses to the hardware. DOS is typically extended using device drivers and Terminate and State Resident (TSR) programs, both of which modify the operating system data structures by intercepting interrupts. Since security was not a concern for DOS developers, even the simplest forms of object reuse are not prevented (e.g., memory and disk are not cleared before allocation). As a result of these weaknesses, viruses and other forms of malicious software have flourished.

3. C2 Requirements

The TCSEC functional requirements for Class C2 fall into four areas:

1. Identification & Authentication (I&A): users must be identified (typically by providing a user ID) and authenticated (typically by providing a password) before using the system.
2. Discretionary Access Control (DAC): Access to objects on the system by subjects must be mediated by a DAC policy based on user and/or group identities.
3. Audit: Auditing of security relevant events must be performed.
4. Object reuse: Objects must be cleared before reuse by subjects belonging to a different user.

The objects to be protected on a DOS system are conventionally defined as I/O devices (e.g., serial and parallel ports) and the files on the disk⁷. Memory and registers are not protected, because DOS systems are single-user at a time. Rather, all transient memory (including memory on peripheral cards) is cleared between each user session.

There are several commercial products for DOS which meet the C2 *feature* (I&A, DAC, audit, object reuse) requirements, including Assure from Cordant and Watchdog from Fischer International. Meeting the C2 *assurance* requirements is much more difficult, since DOS does not have anything which approximates a Trusted Computing Base (TCB) which can be used to ensure that the functions are not being bypassed or otherwise manipulated. Without a TCB, malicious software can disable the security software simply by overwriting appropriate parts of memory. Security remains intact only so long as the implementation remains unknown: the dreaded "security through obscurity".

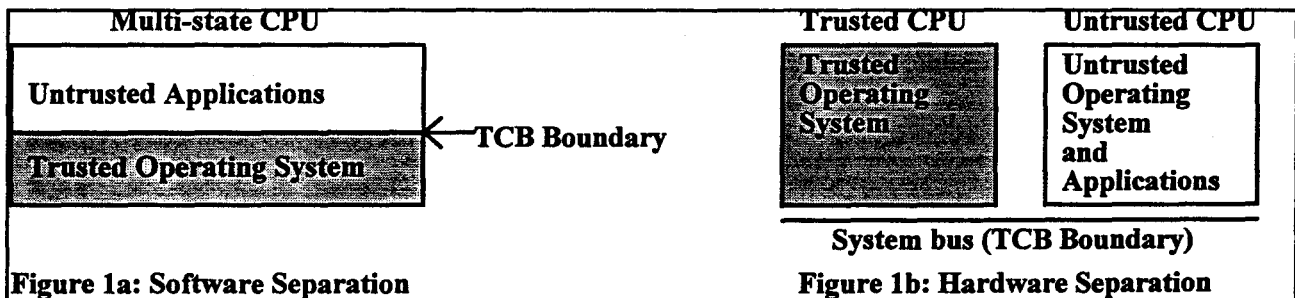
4. Possible Architectures

In this section we describe a number of possible system architectures for a Class C2 DOS system. We divide the approaches in several different ways:

- Separation of TCB and non-TCB using hardware or software. Hardware separation would place the TCB on a physically separate computer, while software separation will run both the TCB and non-TCB on the same processor, using existing hardware mechanisms (such as rings and memory protection) to protect the TCB.
- Granularity of objects protected and access control policies. As was previously mentioned, objects on a DOS system are normally considered to be the files and I/O devices. Using coarser-grained definitions of objects (e.g., entire disks rather than individual files) allows for some simplifications in the system architecture.

All of these approaches rely on the fact that DOS workstations are single-user at a time, and not multi-user systems⁸. We will assume that the computers themselves are physically protected from tampering. This is a large assumption, since PCs are normally on users' desks and not in a locked computer room. However, it is necessary to assume that the user cannot use monitoring equipment to detect what data is being manipulated.

After considering these basic issues, several architectures which take different approaches to these issues are presented followed by several alternative approaches.



⁷As will be seen later, other definitions of the objects to be protected can have interesting effects on implementations.

⁸There are multi-user DOS-compatible systems, but they are not widely used and we will ignore them for the purpose of this discussion.

4.1. Hardware vs. Software TCB Protection

This section compares the hardware and software TCB protection mechanisms. In both cases we assume that the TCB will provide the required C2 features: DAC, I&A, audit, and object reuse.

Software TCB protection schemes use memory management and multi-state features to partition the computer into two or more domains, one or more of which contains trusted code, and one or more of which contains untrusted code (Figure 1a). This is the traditional approach used in building trusted systems. To the best of our

Hardware TCB protection schemes rely on having two (or more) CPUs. One or more of the CPUs run TCB code, and one or more of the CPUs run untrusted code (Figure 1b). The system runs two complete operating systems: one in the TCB portion of the system, and a second (which may be the same or different) in the untrusted portion. The separation mechanism is the physical connection between the machines, which can be viewed as a network. All applications, including Windows, run on the untrusted CPU(s). Because DOS is a single-user at a time system, the complete

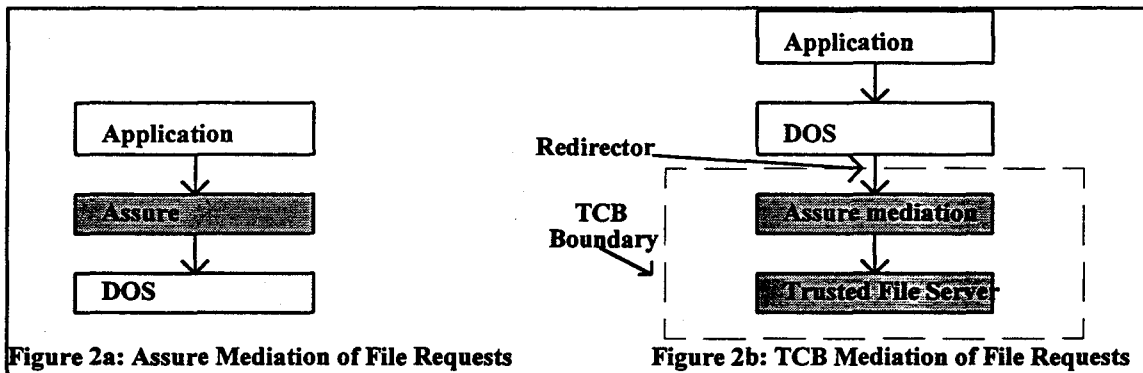


Figure 2a: Assure Mediation of File Requests

Figure 2b: TCB Mediation of File Requests

knowledge, every operating system successfully evaluated against TCSEC used this approach.

In the DOS environment, software protection schemes have limitations. As previously noted, the 80x86 architecture provides for virtual 8086 environments, but not virtual 80386s⁹. Thus, a Virtual Machine Monitor (VMM) [5] can be constructed to provide virtual 8086s. However, any application (such as Windows and many third party memory managers) which believes it has control of an 80386 will be unable to run¹⁰. As a result, the system might be less desirable to users.

contents of the untrusted CPU can be cleared¹¹ between users to avoid object reuse problems.

Hardware separation is implemented by building a board which plugs into the existing personal computer. The add-in board has a CPU, memory, and support chips (e.g., clocks, boot ROMs). It may also include I/O devices, as described in the next section.

There are numerous advantages to a hardware separation approach. Perhaps most importantly, hardware separation is inherently more secure than software separation. Compatibility problems are minimized because the untrusted operating system and applications have total

⁹Virtual 80386 processors could be emulated or provided by single-stepping, both of which are very slow and hence not feasible.

¹⁰A limited version of Windows, known as Standard Mode, can be run in a virtual 8086.

¹¹We assume that a trusted mechanism is possible to clear the untrusted CPU. In the worst case, this can be done by requiring that the computer be power-cycled between user sessions.

control of a CPU, just as they do at present. The add-in board could be used even in low end systems (e.g., 8086 or 80286 computers) which cannot support the multiple domains necessary for software separation. If the protocol is carefully defined, it may be possible to build a hardware separation mechanism which is independent of the operating system being run on the untrusted CPU¹². However, the tradeoff is cost: at least one extra CPU (with its own memory and support logic) must be added to the system. Because of the additional hardware cost, this approach might reduce the potential market for such a product relative to an approach which does not require a second processor. While hardware separation of a TCB has been used in operational systems, it has never been used in an operating system evaluated against TCSEC.¹³

4.2. Granularity of Objects Protected

The objects in a system should be defined in a way that makes sense for how they are used. For DOS systems, the two main types of objects are files and devices. Memory and registers are typically not objects because they belong to the currently logged in user, which is the only subject in the system.

4.2.1. File Access Controls

While DOS does not provide any access controls, third party products such as Assure implement a discretionary access control policy for DOS files. In Assure, the administrator can place access control lists (ACLs) on files, directories, and paths. As an adjunct to the traditional DAC policy, anti-viral features in Assure can be used to deny write access to

¹²The TCB card could have a variety of bus adapters. This would allow building C2 versions of other systems, such as the Apple Macintosh, at a relatively nominal engineering cost.

¹³Using a separate hardware device for the TCB was used in at least one network evaluation (the Verdix VSLAN), but has not been used in any operating system evaluations.

executable files (such as .EXE, .COM, and .BAT files). Assure provides auditing on file operations. The Assure DAC policy meets the C2 criteria, and could be implemented as part of the TCB.

The current Assure product works by intercepting file requests, mediating them, and, if acceptable, passing them on to DOS to be fulfilled (Figure 2a). Assure does not have a non-bypassable TCB, and hence does not meet the C2 requirements. Rather, the file requests must be intercepted and passed off to the TCB where they are mediated and fulfilled (Figure 2b). The access control database must itself be safeguarded by the TCB. In DOS there are several levels at which the intercept can occur. The most useful level is the *redirector*, which is intended to support remote file systems. Thus, the TCB boundary for file operations can be defined as the redirector interface, provided that the TCB is built in such a way that there is no other means to gain access to files.

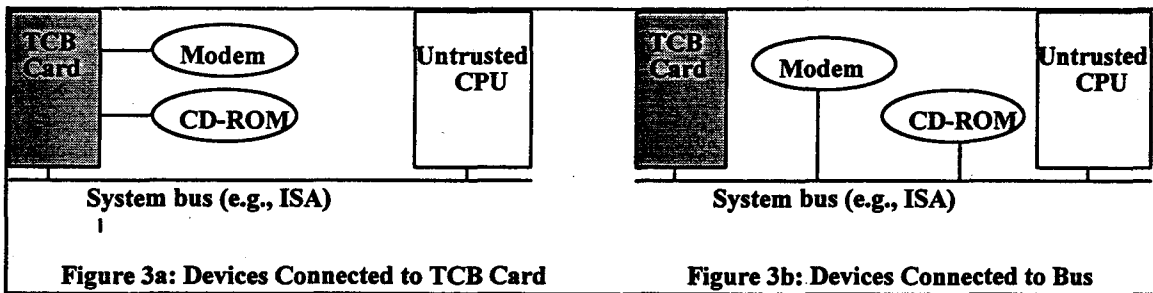
Rather than considering individual files and directories to be objects, another view is to consider disk partitions (known as *volumes*) to be the file objects. In this model, the operating system and applications might be placed in a partition which would be read-only, while individual users would have their own partitions. Sharing of files would be accomplished by putting them in a partition which is accessible to others. In this model, auditing could be performed upon the first access to a file in a volume, but need not refer to the specific file being accessed¹⁴. The main advantage of this approach is simplicity: the access control decision is simply a matter of comparing the sector number being read or written to a range of valid sector numbers. The access control decision could easily be made in hardware, thus reducing the complexity of the

¹⁴Since the audit log would only provide the volume identifier rather than the file name, the log would be of limited value to an auditor.

TCB software and increasing the system performance. While this approach certainly meets the C2 requirements of controlling access to objects, it does not meet user expectations for access control or audit granularity.

A third view of access controls is to view each disk block as an object, and mediate access to the blocks. This is not an unreasonable concept, since DOS provides primitives for accessing individual disk blocks (primitives that are used by many applications). If access controls could be set on each block, this approach could meet the C2 requirements. However, users think in terms of files, not disk blocks, so it is not a particularly useful view.

problem is more difficult. If the devices are physically connected to the TCB (Figure 3a), the TCB hardware could perform mediation. Connecting all devices to the TCB is expensive, because it requires that the physical boards be modified to plug into the TCB card rather than the existing system bus, or that the logic be physically placed on the TCB card. However, if the devices are connected to the existing bus (Figure 3b), the TCB hardware cannot prevent access because the untrusted CPU is free to make requests without any opportunity for mediation. There are at least two ways to avoid this problem: reverse the roles or implement bus locking logic.



4.2.2. Device Access Controls

Just as there are several ways of thinking about files and directories, there are several ways to consider I/O devices. Because of its nature as a single-user machine, all devices on an ordinary DOS system are available to a user with access to the system. DOS applications need not open devices; rather they simply execute hardware I/O instructions to access the devices directly. To meet the C2 requirements we must provide some type of access control on devices. One approach is a single access rights for all devices on the system. It is more useful to provide access control on a per-device basis, so one user might have access to the modem in a system while another user might not.

Implementing a DAC policy on devices is particularly tricky. In a software-separation approach, the operating system can use features of the 80386 architecture to allow or deny access to any device running in a virtual 8086 box. In a hardware separation approach, the

- Reversing the roles means the existing CPU runs the TCB software and the add-in card runs the untrusted software. By doing so, the existing system bus can be made trusted, and the add-in card could have logic to deny access before the request is placed on the bus (i.e., a portion of the add-in card, which interfaces to the existing bus, is part of the TCB). This has the unfortunate side effect that if the add-in card has a lower performance CPU than the existing CPU (which would likely be the case to minimize the cost of the add-in card), upgrading to a C2 solution would reduce the system performance.
- Bus locking logic is the preferred solution: hardware on the add-in card watches the address lines on the bus, and determines whether the address is authorized. If it is not, the add-in card halts the bus (by raising the busy signal), thus crashing the machine! This is a drastic measure, but the only way

to stop an I/O operation which has already begun.

4.3. Example Solutions

This section describes a number of systems which select different solutions to the problems posed in the previous sections. Each of the solutions presume that a trustworthy TCB (whether hardware, software, or both) can be built. The detailed design of the TCB for each case is beyond the scope of this paper; we focus on the system architecture.

4.3.1. Hardware Desktop File Server

In this approach, the TCB is protected using hardware separation. The existing CPU behaves as a diskless workstation. The TCB card includes the disk controller so the disks are physically inaccessible from the untrusted CPU. Serial and parallel ports are part of the TCB card as well so device mediation can be performed without having to crash the system (as previously described).

The software running on the TCB card is primarily a file server, albeit one which services requests only from a single-user at a time. When untrusted software (including the untrusted operating system) needs file access, a message is sent over the system bus requesting the desired access. The protocol used could be NFS¹⁵, NCP¹⁶, or a simpler protocol designed for this purpose. Although the file server software could be a UNIX system, a NetWare server, or any other similar system, minimization of memory requirements (and hence the cost of the TCB card) points to a less sophisticated file server.

Software running on the untrusted CPU is generally unchanged. An untrusted redirector must be added to intercept file requests (e.g.,

¹⁵Network File System, the *de facto* standard for UNIX environments.

¹⁶NetWare Core Protocol, used for communicating with Novell NetWare servers.

open, close, read, write) and send them across the system bus. Virtually all applications, including those which directly manage system memory, can be run. The only applications which will not work are those which insist on directly manipulating the disk, such as Norton Utilities. Programs which access the serial and parallel ports would see the ports if the user is authorized to use them. If the user is not authorized, the TCB board will treat the I/O requests as if the ports do not exist.

The primary advantage of this architecture is the clearly defined TCB interface which has the potential for a high assurance system, while still retaining complete compatibility with existing applications. The primary disadvantage of this approach is the cost to the end user. Depending on the amount of memory required, the card could cost as much as the computer itself.

4.3.2. Hardware Block Mediation

This architecture is similar to the desktop file server concept. However, instead of intercepting *file* requests using the redirector interface, this architecture intercepts *block* requests and forwards them to the TCB. As such, it looks to the untrusted CPU like a disk controller rather than a file server. When a block request is received, the TCB reverse engineers the operation by examining the block number being modified and the type of request. By examining file system structures, it can be determined whether a block is part of a file or a control structure. If the block is part of a file, the TCB can calculate what file the block belongs to and whether the user has the desired access to the file. If the block is part of the file system control structure, looking at the specific modifications (i.e., comparing the new image of the block being written to the old image on the disk) can determine whether a file has been created, deleted, or extended, and the appropriate access control decisions can be made. Attempts to make invalid accesses or modifications can be treated as unrecoverable I/O errors on the disk, thus maintaining the fiction that the TCB is a disk controller.

Dynamic reverse engineering of block requests sounds quite difficult. However, the DOS file system layout is very simple. At least one product¹⁷ has implemented this mechanism without substantial difficulty. Because the system only supports a single user at a time, it is possible to build a bit map of blocks accessible to the user, hence reducing the reverse engineering overhead on each I/O operation.

The main advantage of this approach is that even those applications which access the disk can operate, provided that they don't try to access parts of the disk that are denied by access control restrictions. The main disadvantages are that the TCB interface is difficult to justify (the semantics of the interfaces are very complex) and it is tied to a particular file system layout. The cost of the hardware is probably no less than the desktop file server approach.

4.3.3. Hardware Partition Mediation

This architecture assumes that mediation is done at the disk partition level, not the file level (i.e., user Smith can access anything in cylinders 0-27 and 95-124). Because the mediation is much simpler than any of the other architectures, it could be done by an enhanced disk controller. An add-in board could include hardware to protect devices, but need not have its own CPU. In this approach, the existing CPU would initially run trusted software. After initializing the device protection and partition information, the CPU would run untrusted software. Hardware features would prevent modifying the device protection data until the system is rebooted.

Unlike the other hardware-based architectures, this one could be built fairly cheaply. Without

¹⁷The Arbiter, from Automated Answers, Inc. uses this strategy for a completely different purpose: connecting DEC Q-Bus and UniBus devices to PCs running DOS, Windows, or OS/2. To the DOS machine, a Q-Bus or UniBus disk looks like a DOS file system although it may really contain a VMS file system.

the need for a second CPU and its memory, the add-in board could be very inexpensive. The untrusted software would not change at all, and the trusted software would be minimal. By mediating at the partition level, any untrusted operating system can be used. The tradeoff is flexibility: access control is at a partition level which provides minimal flexibility to an administrator or user.

4.3.4. Trusted DOS

This architecture is a software attempt to make DOS into a trusted operating system. Rather than defining the TCB interface as file requests (and running DOS outside the TCB as a diskless workstation), it tries to define all of the interfaces used by DOS applications and define their security implications. It is architecturally similar to today's C2 and B1 UNIX products: an existing operating system is enhanced, documented, and tested to meet the criteria. However, it is much more difficult to do for DOS, both because there are so many interfaces and because the lack of anything approximating a TCB has encouraged applications to bypass the operating system. Thus, although it is the most straightforward approach conceptually, it is the most difficult to achieve and the most likely to have compatibility problems with existing applications.

4.3.5. Software Desktop File Server, Block Mediation, and Partition Mediation

These architectures are the same as the hardware desktop file server, hardware block mediation, and hardware partition mediation (respectively), except that instead of using two CPUs, a Virtual Machine Monitor (VMM) is used to separate the untrusted software from the trusted software. The untrusted domain believes it is running on a diskless workstation (just as the untrusted CPU did in the hardware architectures), but instead of making file or block requests to a TCB card, it makes the requests to a data server in the same CPU. The VMM enforces separation and passes messages between the untrusted and trusted software, but

provides no facilities on its own. Note that there are three operating systems running on the same CPU in this architecture: the VMM, the trusted operating system, and the untrusted operating system (Figure 4).

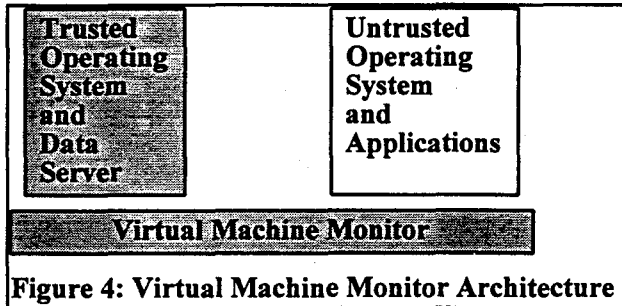


Figure 4: Virtual Machine Monitor Architecture

The advantage of these approaches compared to the hardware versions is that they do not require a second CPU with the attendant costs. The disadvantages are that they will not be able to run applications (such as Windows) which require complete control of the CPU, and that performance will suffer because of the additional overhead created by the VMM and the data server.

4.4. Other Approaches

This section describes several other alternative architectures which do not fit in the general scheme described above.

4.4.1. Device-less Workstation

Most of this paper has discussed means to provide controlled access to devices and files on a workstation. Another possibility is to make not merely *disk*-less workstations, but device-less. In this approach, workstations would have a CPU, memory, keyboard, mouse and screen, but would not have any floppy or hard disks, or other I/O ports. A modified network card would not allow promiscuous mode (which allows a workstation to listen to all messages on the network). Procedural controls could be used to ensure that the computer is power cycled after each user is finished, thus clearing all memory on the machine. Trusted software would be remotely booted from a file server, and all authentication would be performed by the file

server, *not* by the workstation. As a result, the network must be physically protected from eavesdropping. Access to files could be via a network, which could enforce access controls instead of trying to enforce access controls at each workstation.

This is clearly the simplest approach to providing a secure workstation, as the workstation becomes an intelligent terminal. Just as dumb terminals can be used in trusted mainframe and minicomputer systems, so could a device-less workstation be used in a network environment. The clear tradeoff is usability: if the central file server is unavailable, all workstations are useless. As such, while we feel that such an approach could meet the C2 requirements, it would not be particularly useful.

4.4.2. Plug-In Chip

One of the more unusual architectures is to perform mediation at the CPU level. In this approach, the existing CPU would be removed from its socket, an intermediate chip set in place, and the original CPU plugged back on top. The intermediate chip would be the TCB and would have the capability to mediate each operation.

There are two major problems with this approach: it may not be possible to do the physical operation and obtaining assurance from the design would be impractical. Many chips are soldered to boards, so they would not be able to work with this approach. Also, there are many different types of CPU chips (e.g., number of pins, clock speed), and each would require a different mediator chip. It is far from obvious how to design a mediator chip which would enforce security policies such as DAC and auditing. For these reasons, this approach is mostly an intellectual curiosity.

4.4.3. Hosting DOS on Other Operating Systems

Perhaps the most obvious solution to building a trusted DOS workstation is to run DOS hosted on another system. For example, there are

several UNIX systems being evaluated at C2 or B1 which include DOS emulators, and Microsoft is currently evaluating the Windows NT operating system which also includes a DOS emulator. IBM's OS/2 can also emulate DOS, and it appears capable of meeting the C2 requirements. All of these are both reasonable and unreasonable: they would certainly make the task of building a C2 system simpler (it's much easier to build a C2 UNIX or OS/2 system than a C2 DOS system), but all carry heavy penalties. While DOS can be run in 1MB of memory, and DOS with Windows can run in 2MB, current UNIX, NT, and OS/2 operating systems all require at least 6MB (and preferably much more). Disk space requirements for UNIX, NT, and OS/2 are also much greater than DOS. Thus, the cost to the end user is significant in terms of additional memory and disk space required. Additionally, the end user must learn to handle another operating system to get their job done.

The second main difficulty with running DOS hosted is that none of the emulations are complete. Some applications simply won't run. Windows applications tend to be relatively non-portable. While the introduction of WABI (Windows Application Binary Interface) may eventually yield the ability to run Windows applications compatibly on non-Windows systems, for now this seems to be a goal rather than a reality.

5. Conclusions

There are many ways to provide C2 DOS systems. The basic techniques have been known for years: building virtual machine monitors, trusted operating systems, and hardware separation. Because of the engineering realities of DOS and the massive installed base, compatibility with ordinary DOS is critical for a product to have a chance in the marketplace.

Unfortunately, none of the solutions are perfect. The hardware separation solutions are relatively expensive to the end user. Software separation solutions suffer from compatibility problems

due to the lack of a completely virtualizable processor. Performance is a likely bottleneck in the software solutions as well. Trusted operating system approaches may have the greatest opportunity for compatibility, but would be the hardest to gain assurance.

We do not believe there is a single "best" solution. Choosing the appropriate solution is a matter of balancing compatibility, performance, and price requirements with engineering effort available.

6. Acknowledgments

The authors are grateful to other members of the Cordant development team who made many useful comments and suggestions on this article, including Ron Field, Mike Hantman, Rod Roark, Barbara Maguschak, and Mike Ryan. We also appreciate numerous conversations with our colleagues at Novell, including Craig Teerlink, Kevin Kingdon, and Doug Hale.

7. References

- [1] *The Programmer's PC Sourcebook, Second Edition*, Thom Hogan, Microsoft Press, 1991.
- [2] *Undocumented DOS: A Programmer's Guide to Reserved MS-DOS Functions and Data Structures*, Andrew Schulman, et. al., Addison Wesley, 1994.
- [3] *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1995.
- [4] *DOS Protected Mode Interface Specification*, Version 1.0, The DPMI Committee, March 12, 1991.
- [5] *Operating Systems*, Stuart Madnick and John Donovan, McGraw Hill, 1974.

A Practical Hardware Device for System and Data Integrity as well as Malicious Code Protection

T.E. (Ted) Elliott

Communications Security Establishment
Department of National Defence
P.O. Box 9703, Terminal
Ottawa, Ontario, Canada
K1G 3Z4

613-991-7506 Fax: 613-991-7411

telliott@cse.dnd.ca or, TEElliott@dockmaster.ncsc.mil

Abstract

This paper outlines an applied research project with an objective to develop a practical integrity product implemented in hardware. The analysis and rationale involved in this prototype development are presented. Areas where such hardware devices may be considered for integrity protection are presented.

Key Words: National security policy, computer security, integrity policy, hardware device, read-enable, write-enable, disconnect, anti-virus, key-lock, storage access control, storage device, interface, mode of operation, intellectual property

Introduction

Many authors have provided a great deal of discussion of the theoretical concepts and alternative views on what precisely the term *integrity* should imply, both for “data” and for “system” attributes and the respective protection philosophies.^[1 - 6] Little discussion or guidance for integrity protection is available within federal government doctrine¹ either within the United States or Canada. Although there is mention of integrity within the TCSEC^[7] it is only within the confines of the trusted computing base: “System Integrity – Hardware and/or software features shall be provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the TCB (Class C1).” In spite of the major expansion of the CTCPEC^[8] to include *integrity* criteria, there has yet to be any product offered for integrity consideration regardless of whether confidentiality or availability forms part of the same product security policy. Considering the supplementary cost and “fuss factors” of traditional “trusted products”, it was felt necessary to keep an open approach, if possible, to include any integrity product development that might also be used with all commercial operating system platforms.

At the NIST/NCSC 1993 16th National Computer Security Conference two items are important. Mr. Pickering’s keynote address stressed the importance of developing cost effective security products through international cooperation. As well, the need for an integrity model, for electronic voting systems and other systems where life or safety issues are paramount protection requirements, was presented.^[9] A prototype software program was proposed to properly identify

¹ Doctrine is used to cover security policy, standards and guidelines.

executable software on the IBM PC platform and this relates somewhat to system operational integrity.^[10]

In summary, there are no specific ways for addressing integrity from within the Canadian and U.S. federal government doctrine now available— no defined integrity labels, tags or integrity process controls are available. At present in the U.S. “No government wide schemes exist which are based on the need to protect the integrity or availability of information.”^[11] However, the same paper includes “Interpretation of the Computer Security Act’s definition of sensitive is, ultimately, an agency responsibility. Typically, protecting sensitive information means providing for one or more of the following:

...
Integrity – The information must be protected from errors or unauthorized *modification*.
...

Another Bulletin in this series from NIST concerning security issues in public access systems^[12] lists specific security issues:

- #1. - Maintaining Data Integrity ...
- ... #5. - Preventing Computer Viruses ...
- #6. - Legal Considerations ...

The first, maintaining data integrity, includes mention of using CD-ROM technology for on-line data storage for distribution with physical protection from user modification. Under discussion of virus prevention there is no mention of any other hardware technology. For guidance on legal issues operators are urged to consult legal counsel.

There are too few hardware products, with integrity features, presently available in the *client/server* or desktop environment. Previous work involving a combination of software and hardware functionality was uncovered in the course of this work, but was not evidently available in commercial production.^[13]

Background and Analysis

When any computer system is required to be secured and examined, for purposes such as audit, or for a police expert’s forensic court evidence, how can the integrity of any information residing upon storage devices be assured? These platforms, in most cases, also comprise non-trusted operating systems. Upon power-up, how can the integrity of their total information set, retrieved from write enabled storage, be adequately assured?

A number of years ago, a project database system within the federal government was running at two separate sites, utilizing duplicate application environments and database files. During a backup at one site, operator error unknowingly damaged the hard disk before the diskette backup was started. Murphy was the operator’s best friend because only one set of diskettes were used and these were overwritten with the backup of the severely damaged hard disk volume. A panic call was received requesting a complete set of the database files from the second site. Seven DOS diskettes and fifty-six files were copied and rushed by taxi to the second site for restoration of the database by the application programmer. Fifty-four of the files copied without error to a

PC with a replaced hard disk, but two files could neither be recovered by DOS nor with Norton Utilities™. Immediately, the security officer had good reason to ponder the integrity of the sole remaining database! As it turned out, a second copy of the two suspect files was successfully used to complete the restoration of the damaged site's operations – but what do we really have even today for maintaining “integrity” with any database system, over any operating system? Further, not even the expert application programmer knew that the DOS *verify* command should have been set to “Verify ON” for all operations. It seemed to me that even with this installed, a lot of third party software would not utilize “DOS verify on” when backup to tape was employed.

During the second week of February, 1994, Sybase, one of the vendors of Trusted database products, currently undergoing evaluation with the NCSC of NSA were making a product announcement and demonstration in Ottawa, the capital of Canada. Based upon my questions with individual staff of this particular company, there appeared to be some interest in considering feasibility of adapting their database software to not only reside behind read-only hardware protection, but to also consider allowing storage volume(s) of data also to reside behind such protection, when production requires only read access for most users.

Since 1985, incidents of malicious code interruption or data loss to federal processing systems have been reported at an increasing rate.² And, this is in spite of an investment which I believe includes “anti-virus” and related scanning and virus removal software that has cost the taxpayer millions if not billion(s) of North American “dollars”! Now, I believe there has never been a malicious code incident involving the standard facsimile system, at least those that do not employ computer processing and storage devices. In addition I've often heard system security colleagues say to me “If the product is only implemented in software, it's probably not strong enough.” They would say this to me with the TCSEC and the CTCPEC (criteria) objectives having been held close to their hearts (and mine). Ancillary to this of course was the corollary statement “If it requires reasonably strong “trust” the product would employ a hardware basis.”

In 1986 I received an 8086 Intel based IBM-compatible PC XT complete with a 20 Mb hard disk. Within two years I was planning the addition of a modem to allow me to link this “private” host to those available on dial-up lines. Being aware that some “public” bulletin board systems, the Internet, and related environments may be plagued with malicious code and “unauthorized actions”, I was concerned about my own personal reputation, i.e. integrity, should any of my PC information become tainted with any malicious code – criminal action, modification etc. In signing for authorization of my access rights to Internet hosts at CSE, called *manitou* and at NSA, called *dockmaster*, I felt apprehension with particularly my own hard disk storage content and its “integrity” over which I had not even had reasonable configuration controls or related tools. Furthermore, off the shelf software from the local KMart might contain anything — it didn't even run when loaded! What I would have liked was the ability to boot the PC as a “dumb” terminal, with only basic local printing and communications software protocol, and occasionally down/upload with diskette(s). No write capability was desired from the whole hard disk storage during such remote access.

² In Canada, the Royal Canadian Mounted Police EDP Security Branch have the responsibility to assist all federal institutions except for the Department of National Defence, and this includes a special team of experts for assisting them on request concerning malicious code incidents.

Based upon the above experience as a starting goal, I set about planning to define what might be within possible grasp. As well my objective was to achieve a product that might help the widest possible domain, and which coincidentally may be novel enough to warrant granting of intellectual property rights. Figure 1 presents my view of those relevant parts of the PC platform, as one example. The figure of course is applicable to any other platform such as offered by SUN, Hewlett Packard, Digital Equipment Corporation and the many other manufacturers.

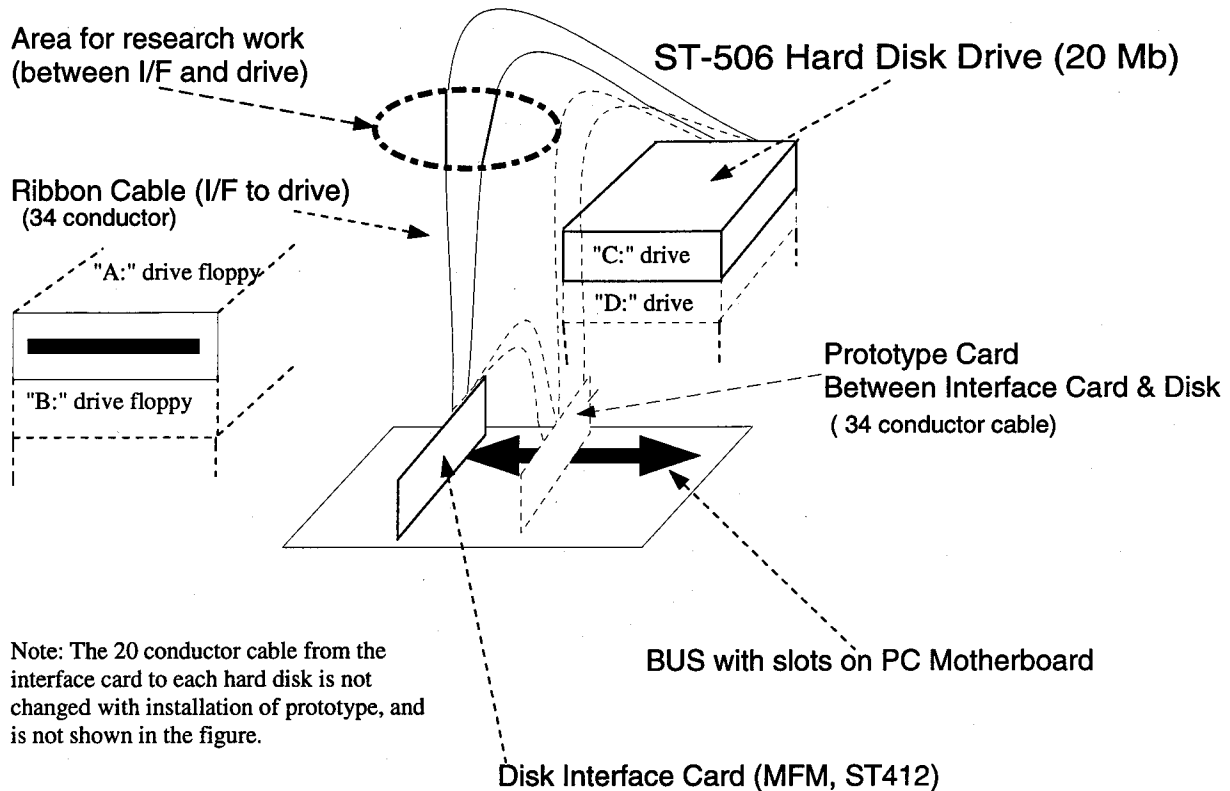


Figure 1 - Locating the possible Hardware Control Locations

In view of the design shown there would not seem to be much that can be done without major cost implications within the motherboard of the PC design. Nearly all of the CPU processing and memory chip technology is purged upon power off, or reset, and nearly all of both the DOS operating system and application software is loaded from the hard disk at either power up or re-boot. Now when the loading of the CPU and motherboard components occurs at power up (or re-boot) the storage device must function in read-only fashion, but in the case of magnetic disks, such as the 20 Mb model of the present platform, it was, and still is, designed to be only operable in the combined "read and write enabled" fashion. This totally exposes the disk as a "wide open" vulnerability to any conceivable threat agent, malicious code, unauthorized activity or natural accident! My personal goal therefore was to provide a security (special operation) control for the specific mode of connection between any/all system storage devices and the CPU or motherboard. By accomplishing this goal I would attempt to allow improved control through:

1. installing new software or operating system or data files with "read & write enabled" mode;

2. changing appropriate disk volumes to lock them in “read-only enabled” mode;
3. changing appropriate disk volume(s) to lock them in disconnect (“read & write disabled”) mode;
4. forcing object reuse by ensuring power off sanitization before the mode change is effected;
5. reporting to the operator, any attempt to conduct a write or read operation that was not permitted by the current operational mode (for each disk).

Indeed, the floppy drives provide some of the above functions already. Reference to the various types of information is also appropriate at this point. Figure 2 shows at least three various types of information resident within the storage device(s) of any system.

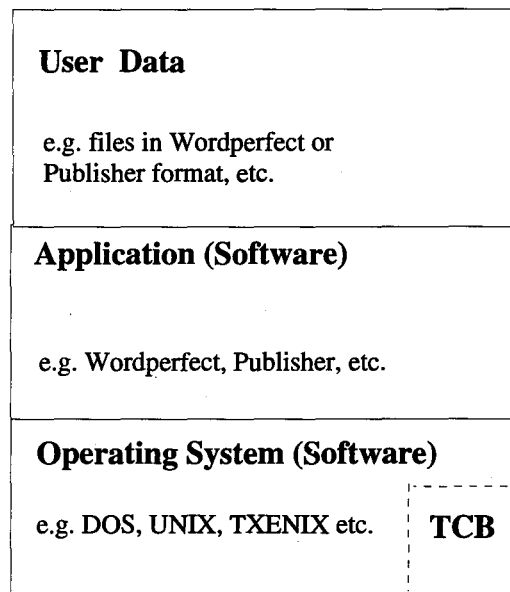


Figure 2 - Types of Information

All of these types of information are stored on one or more computer storage devices connected to the central processing unit or mother board, in the typical PC. It was therefore expected that all could be protected for integrity purposes if a single device could be conceived!

Accordingly, planning and design was done for the following hardware prototype.

Prototype

Purpose

The prototype (Hardware Protection Control for Computer Storage Devices) provides user control of read and write access to each connected storage device(s). The primary intent was to achieve **a hardware only** (i.e. no software is involved) implementation of the following *device security policy*: User, or a special security–authorized user is provided absolute control of read and write action to all storage devices by a physical switch, in which only hardware would be used. No software or firmware was desired for the avoidance of increased risk and evaluation costs.

First Implementation

The prototype device was made for the IBM PC XT/AT and IBM compatibles that use the ST506 disk / ST412 disk interface standards. The prototype design was based upon a half-card, with certain Tempest design considerations, and an extra ribbon cable to insert it between the existing system disk controller and the physical disk drive(s). Component costs were approximately \$100 (without Medico lock). Alternative designs may be easily implemented for other storage device interface standards e.g. SCSI, IDE, etc.

Functional Description

The device as prepared offers three different modes of hardware operation, *selectable by physical switch*³:

1. **Normal** Operation with both read and write enabled;
2. **Protected** operation with only read enabled but write disabled;
3. **Disconnected** operation with both read and write disabled;

The selection switch may be an external toggle switch, an externally mounted Medico keylock, an internal keylock or by internal DIP switch. To prevent any "object reuse" or contamination when a new mode is selected, the user may require system reset or cold boot prior to mode change being effected. Present device mode setting indication may be provided by:

- a. coloured LED e.g. (Position 1. Green, Position 2. Yellow, Position 3. Red)
- b. by other suitable means, such as switch position labels.

When any error occurs an option to allow selection of:

1. reporting the error back to the system; or,
2. preventing the error from being reported back to the system may be user setup.

When an attempt to read or write occurs that is not allowed (in the present mode) then:

1. a sound annunciator beeper (with status active or inactive); or,
2. notice by other suitable means may be provided (e.g. light indication).

There is also need to allow the device to control multiple devices, without restriction on the mode that each may be running in or be changed to.

Applications

Some typical installations may include:

- a. With the switch in the **Protected** read-only position the system storage is immune from all modification, malicious code, viruses etc.

³ A fourth mode may be feasible i.e. Special operation with write enabled but read disabled.

- b. With the switch in the **Protected** read-only position in a sensitive text word processing application (e.g. on a hard disk containing MS-DOS and Wordperfect), all sensitive e.g. U.S.: Unclassified but Sensitive, or Canada: Protected-A, Protected-B or higher sensitive data is absolutely restricted to other storage, guaranteeing the hard disk cannot ever contain or receive such sensitive data and cannot float up to the "high water level of sensitivity". The storage device will remain "Black" regardless of "Red" information processing at any level or category of sensitivity to disclosure.

- c. With the switch in the **Protected** read-only position on a hard disk containing any trusted system or secure application software (e.g. financial and banking functions), the information integrity is and shall remain absolute.

- d. With the switch(s) **Disconnected** removing read and write MS-DOS system access, and allowing read-only access to MS-DOS boot and communication software (e.g. terminal emulation of dumb terminal mode for remote trusted host workstation access) the PC is reduced to the minimum for trusted system access; integrity of the terminal emulation setup is guaranteed by Read-Only mode while the more powerful MS-DOS system is removed (no read or write).

- e. With storage device(s) secure in the **Protected** read-only mode, for software, including trusted tools, used for development of trusted products, one can be assured of their maximum integrity in a fashion parallel to the intent expressed in the "Yellow Books"^{14, 15} for a "closed environment".

The prototype wire-wrap half card is shown in Figure 3. This card has been tested in a PC with live versions of selected malicious code and in all cases of use in both Protected and Disconnected mode of operation, the connected disk storage integrity was untouched. A commercial version was requested for installation of the device in an office virus scanning workstation, but at the time there was not any commercial design ready for such production. With the passage of time, the MFM interface has been phased out on all current PCs in favour of the SCSI and IDE interfaces.

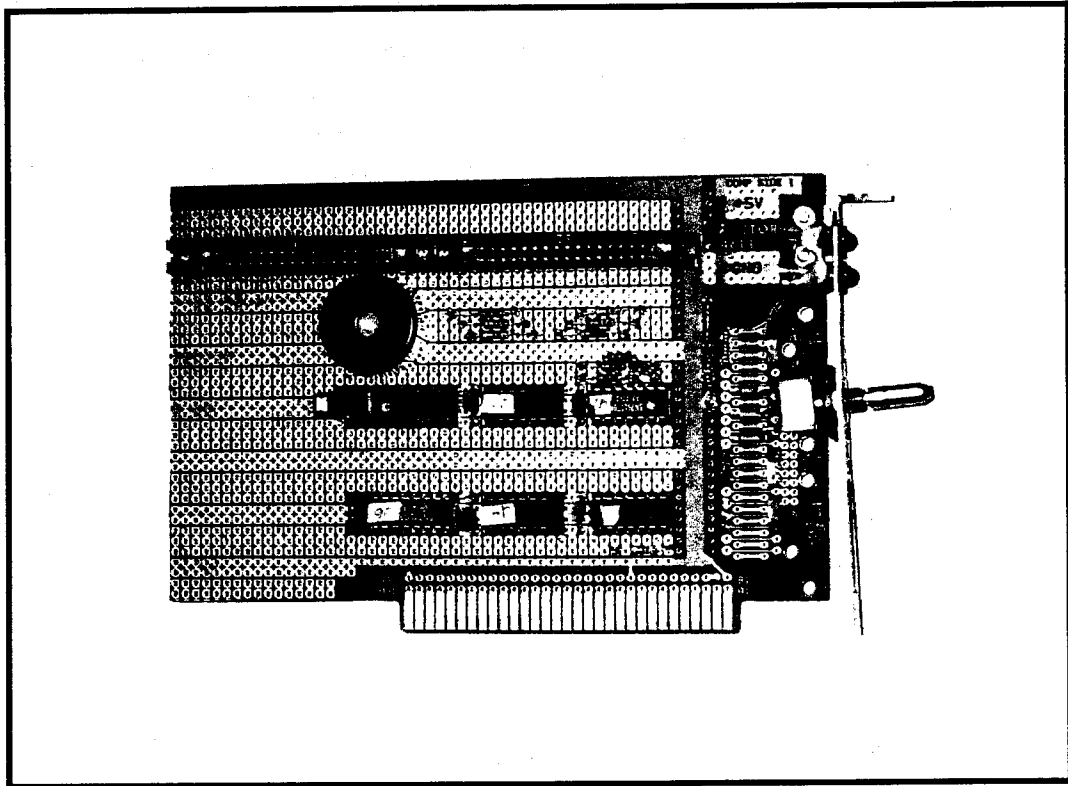


Figure 3 - Prototype on a half card for a personal computer

Summary

This product prototype may assist in the refinement of criteria for integrity. Consideration of such a prototype under the Federal Criteria, under the CTCPEC, and under the Common Criteria to be released this spring may allow its basic product security policy to be validated or lead to appropriate prototype changes. This product is also available for commercial implementation should there be demand to allow this practical security to be realized, in its present form. Any readers or vendors who may be interested in commercializing this development are invited to enquire about a license agreement from Mr. K. Aspila, Director of Patent Administration, Tel. 613-992-3800, Fax 613-995-5111, Department of National Defence, MGen George R. Pearkes Building, 101 Colonel By Drive, Ottawa, Ontario, Canada, K1A 0K2.

Acknowledgments

The author is grateful for extensive training provided by the Communications Security Establishment including legal encouragement from Mr. Reagan Walker of the Department of Justice and especially to Messrs. R. Gonzalez, R. Hysert and T. McKenzie for their contribution of hard work, support and encouragement. Mr. H. Kelly of the Department of Defence contributed significantly on this IP application.

References

1. K. J. Biba, *Integrity Considerations for Secure Computer Systems*, Mitre TR-3153, The Mitre Corporation, Bedford, Massachusetts, April 1977
2. David Bonyun, I.P. Sharp Associates Ltd., Ottawa, Canada, *A New Model of Computer Security with Integrity and Aggregation Considerations*, Department of National Defence, Ottawa, Canada, Contract Report TP-78-4116-1, 21 March 1978
3. David Bonyun, I.P. Sharp Associates Ltd., Ottawa, Canada, *Aspects of Integrity*, Contract Report CP-86-5402-13, March 1986
4. *Report of the Invitational Workshop On Integrity Policy in Computer Information Systems (WIPCIS)*, October 27-29, 1987, Bentley College, Waltham, Massachusetts, sponsored in part by the National Bureau of Standards
5. Zella G. Ruthberg, William T. Polk, *Report of the Invitational Workshop on Data Integrity*, NIST Special Publication 500-168, September 1989 (377 p.)
6. Terry Mayfield, J. Eric Roskos, Stephen R. Welke, John M. Boone, and Catherine W. McDonald, Institute for Defense Analysis, *Integrity in Automated Information Systems* NSA/NCSC C Technical Report 79-91, 16 September 1991 (149 p.)
7. Department of Defense, *Trusted Computer System Evaluation Criteria, (TCSEC, "Orange Book")*, DoD 5200.28-STD, December 1985, 13
8. Communications Security Establishment, Government of Canada, *The Canadian Trusted Computer Product Evaluation Criteria*, Version 3.0e, January 1993 (available by FTP from: [ftp.cse.dnd.ca](ftp://ftp.cse.dnd.ca) login as *anonymous*, password: *username@site*)
9. Roy G. Saltman, *An Integrity Model is Needed for Computerized Voting and Similar Systems*, Proceedings of the 16th National Computer Security Conference, September 20-23, 1993, 471-473
10. Russell Davis, *Software Checking with the Auditor's Aid*, Proceedings of the Sixth Annual Computer Security Applications Conference, Tucson, Arizona, December 3-7, 1990, 298-303
11. U.S. Department of Commerce, Technology Administration, NIST, *CSL Bulletin - Advising users on computer systems technology*, November 1992, Sensitivity of Information, page 2, paragraph 2.
12. *ibid*, May 1993, pages 3-5.
13. Datapro Reports on Information Security, pages IS32-338-101 to -104, © 1989 McGraw-Hill, Incorporated
14. DoD Computer Security Centre, *Computer Security Requirements - Guidance For Applying The Department of Defense TCSEC In Specific Environments*, CSC-STD-003-85, 25 June 1985, "Yellow Book"
15. DoD Computer Security Centre, *Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements - Guidance For Applying The Department of Defense TCSEC In Specific Environments*, CSC-STD-004-85, 25 June 1985, "Yellow Book"

PARTITIONING THE SECURITY ANALYSIS OF COMPLEX SYSTEMS

Howard Holm
National Security Agency
ATTN: C71
9800 Savage Road
Fort Meade, MD 20755-6000
(410) 859-4458
Holm@DOCKMASTER.NCSC.MIL

This paper re-examines the problem of Trusted Computing Base (TCB) boundary definition with specific application to systems built from multiple components. Although the classic definition can be expanded in a relatively straight-forward way, the composition of components is rarely considered formally early enough to guide the product design. The concepts should be useful in product development, evaluation, integration, evaluation, and accreditation

Keywords: distributed systems, networks, TCB, TCSEC, TNI, TDI

June 3, 1994

Over the past several decades, the most common information system architecture has progressed from isolated single-vendor solutions to highly integrated combinations of hardware and software from independent developers. The foundations of information system security were initially constructed with the early isolated single-vendor types of systems as models. The classical foundations can be extended to encompass the newer more complex architectures. However, there are few successful implementations which can be emulated. The goal of this paper is to provide a re-statement of information security fundamentals with respect to the more complex architectures. It is hoped that this will serve to provide a framework for discussion of security between developers, integrators, evaluators, and accreditors.

Background

As described in the Trusted Computer System Evaluation Criteria (TCSEC)[2], the Department of Defense (DoD) has a long history of research in computer security. Research by itself is insufficient to increase DoD security; the findings of the research must be reflected in available computer products.

The National Security Agency (NSA) Trusted Product Evaluation Program (TPEP) evaluates the security of computer products, and thereby provides a source of computer security advice and an incentive to produce ever more secure products. The primary metric used by the TPEP to assess security is the TCSEC, which is focused on stand-alone, multi-user, single-vendor systems. As systems have grown more complex, the TPEP has attempted to address the security issues with interpretations of the TCSEC such as the Trusted Network Interpretation (TNI)[4] and Trusted Database Interpretation (TDI)[5]. The former was originally intended to address systems constructed from products that individually could not address all the TCSEC requirements. The later was intended to address applications layered on an existing TCSEC system.

It is the author's experience, as an evaluator for the TPEP, that many individuals fail to recognize the value of the systematic approach to systems security principals embodied in the TCSEC, but rather are distracted by the detailed requirements in the TCSEC and its interpretations into believing that none of the TCSEC principles apply to today's more complex technology. The remainder of this paper will attempt to show how those principles can be applied and provide some high-level examples of implementation. This paper does not necessarily reflect current TPEP policy and any perceived differences should not be inferred to reflect changes to TPEP processes.

The Security Policy

The first and most basic principle which can be extended is that of the system security policy. The system security policy includes the confidentiality, integrity, and availability concerns related to the particular information system. It may also include a policy for accountability. This is traditionally embodied in a set of requirements. The TCSEC is an abstraction of many of the policy requirements for a general purpose Department of Defense (DoD) system.

A policy need not be contained within one component of a system. The TNI is an example of partitioning (the TCSEC) requirements into a smaller set of requirements for each component. The TDI offers an alternate way to partition policy components among discrete software components. Although it is implied in the TNI, the TDI presents a more explicit argument that the requirements for each component are not completely independent and some must apply to every component.

Hosmer [3] and others have examined the problems of integrating disparate policies. However, even if the policies of the various components are compatible with a global policy the implementation of the components may be incompatible. This is where many current implementations encounter difficulty. When each product developer determines the security requirements for his specific product, implicit assumptions are made about the policy enforced by other associated products. An integrator is then left to assemble the pieces and try to provide a coordinated policy as an afterthought. The TPEP program in its evaluations against the TNI requires a Network Security Architecture Document (NSAD). A well-written NSAD can be very useful in describing the policy and requirements expected by a component of other components. Some thought is currently being given to requiring a

“Trusted Software Architecture Document” for TDI evaluations that would describe the trusted application interface provided and/or expected by a product.

While the TPEP use of these documents for evaluation is significant, it needs to become standard practice for integrators and accreditors to expect such documents. These documents should be made widely available early in a product’s life-cycle. It does not help an integrator to build a cohesive system to be handed documents describing widely disparate security policies and implementation expectations. Thus even though the components may be enforcing some local policies, the usefulness of the NSAD is limited to those portions of the policies which are being enforced by the components as a whole, and to policies which build upon more primitive policies.

The Security Domain

Once the security policy is established, the system security features which implement the policy must be developed, analyzed and tested for each component and for the combined system. The traditional view is that there are exactly two types of entities; trusted and untrusted. With respect to this the TCSEC does a reasonable job of defining the Trusted Computing Base.

Trusted Computing Base (TCB) - 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.

The “protection mechanisms” here must be seen not only as the mechanisms implementing the security features of the system, but rather everything that can affect the correctness of the security features. The TCB ideally demonstrates the three design requirements for a reference validation mechanism as described by Anderson [1]:

1. must be tamperproof
2. must always be invoked
3. must be small enough to be subject to analysis and tests, the completeness of which can be assured

Historical Definitions

This historical paradigm of the TCSEC is to take the view that a system contains some information, and provides interfaces for accessing that information to system users. A boundary is drawn around the parts of the system that enforce the security policies with respect to that information. This then is the TCB.

A traditional understanding of the TCB as used in the TPEP includes the following:

1. In general, all hardware (and firmware) must be part of the TCB because it operates at a level where it could subvert any security policies or mechanisms implemented in software. In some very restricted cases, it is possible to have non-TCB hardware, even for what are ordinarily TCB func-

tions (such as disk drives), through the use of hardware isolation mechanisms or cryptographic techniques.

2. All the mechanisms needed to meet the security requirements must be part of the TCB. The TCSEC requirements provide a good example of what is needed and is what TPEP evaluations use as the starting point for determining what is in the TCB.
3. The fact that a mechanism is in the TCB does not mean that its only use is in the context of the TCB. For example, commands used by administrators and libraries used by mechanisms within the TCB can be used by non-administrators and need not be considered a TCB interface if the mechanism cannot affect the security policy when called from outside the TCB.
4. Although not active components of the system, the databases on which the TCB software depends must be considered part of the TCB.

This sort of definition is useful for determining what in the system needs to be controlled and analyzed. But it has some limitations even in the historical context.

For some policies, such as Discretionary Access Control (DAC) and object reuse, the only way to ensure that the TCB can always enforce the requirement is for everything a user executes to be part of the TCB. If users are not constrained to execute only TCB code then the code they execute may

1. not invoke the expected TCB interfaces,
2. invoke additional TCB interfaces, or
3. simulate the effect of the expected TCB interfaces.

This is the classic trojan horse style of attack. While TPEP evaluations do impose some restrictions on what a system administrator may execute in an attempt to limit this sort of attack, in general a non-administrative user is not so constrained.

Another historical weakness is in the area of terminals. A subverted terminal could, for example, store passwords and recall them later. This is clearly something that could affect the correct operation of the security mechanisms. TPEP evaluations often assume a "dumb" terminal which has no programmability. The interpretation of "dumb", if taken more seriously, would imply an examinable set of requirements limiting the functionality of the terminal. In a sense levying implementation requirements on the terminal to support the system-wide policy.

Expanded Definitions

As network components have become more prevalent, it has become clear that a terminal is a very simple case of a connected component. In the case of more complex interfaces, an NSAD supplies the security requirements for the missing component. Nevertheless there is, at some level of abstraction, a system with a coherent security policy. Even Internet connected hosts have some sense of the larger policy restricting users to hosts for which they are authorized. In the case of simple hardware (e.g. disk drives and terminals) a short description of the security characteristics of the device would seem

appropriate. The policy they are required to conform to may be simply to be policy-neutral - neither enforcing any policies or impeding any of the larger system's policies.

Multiple Security Domains

Just as the TNI and TDI have allocated policy to different components, they have allocated the TCB to different components called TCB partitions and TCB subsets respectively. While the TCB is required to protect itself from everything outside itself a component TCB is not required to protect itself from the TCB of other components except to the extent the interface descriptions mandate protection.

In using the historical definitions, there are essentially only two domains: trusted and untrusted. This comes from the fact that the TCB requirements are explicit that the TCB not rely on anything outside itself. A more generalized notion is to break up the system interfaces into collections that have similar policies.

Every information system used in critical applications would require the existence of a domain representing the TCB. Special application systems might require nothing other than this domain. More typical is the historical conception of a general purpose computer system. The system itself and administrators are trusted, the users are all trusted with their own data, but not with each other's. In these cases there is also an implied third domain - the outside world - and it is not trusted at all.

In the following examples, the system and administrative domain is depended on by all the others and fills the role of the TCB. While from a TPEP evaluation perspective everything else is simply untrusted with respect to the TCB, in the real world there are gradations of trustiness, and potentially more mechanisms to implement the security policy. At the very least there are hierarchical schemes of trust where some domains are completely trusted by other domains, and peer domains where both peers trust some underlying domain, but not each other. There are also schemes where some domains are trusted only to a certain extent by other domains. This last case is the most difficult to characterize in integration documentation, and at some level of complexity may not be adequately describable.

The following example systems will be used to clarify the multiple domain concept. The term "TCB" will be used to refer to the most basic self-protecting part of the system. It is similar to a traditional TCB, however, in some cases administrative interfaces will not be included. The TCB and the "Outside world" are always domains in the examples.

Example 1: A University Computer

A university computer system can be pictured with four domains.

1. Administrators & TCB- completely trusted
2. Accounting - trusted with student financial information, but not grades
3. Registrar - trusted with all student grade information

4. Educators - trusted with limited student grade information and their own information
5. Students - trusted only with their own information, and not to modify their own financial or grade information
6. Outside World - e.g. hackers, former students

In a system that allowed on-line manipulation of grades a facility (e.g. trusted path) to ensure that only authorized educators were able to manipulate grades would be needed as well as controls on student financial information.

Example 2: A Hospital Computer

A hypothetical hospital can also be pictured with six domains. In fact the domains here share many parallels with the University Computer.

1. Administrators & TCB- completely trusted
2. Supervisors - trusted with all patient information
3. Emergency Medical Staff - trusted with all patient medical information
4. Medical Staff - trusted with information on own patients
5. Accounting - trusted with patient financial, but not medical information
6. Outside World - patients, visitors

It is clearly easier to imagine a hospital environment limiting users to a single application interface.

Example 3: A Utility Company Operations Computer

Perhaps the simplest end of the spectrum. A computer system trusted only to prevent unauthorized users' access.

1. Administrators & TCB- completely trusted
2. Outside World - terrorists, hackers

Example 4: A Networked Development System

This is probably the most typical style of system used today. Not only are user interfaces of interest from a security policy standpoint, but interfaces to other systems are equally of interest.

1. Administrators & TCB- completely trusted
2. Other systems within the department - completely trusted.
3. Other systems within the company - trusted with company wide information
4. Users - trusted with their own information as well as public information, may be physically remote
5. Outside World - trusted to see press releases, etc.

Here the system can not always enforce the security policy on data controlled by the users. However, it is responsible for correctly implementing a coordinated security policy with the systems in the fourth and fifth domains.

Example 5: A DoD Multilevel System

This is perhaps the most interesting example. There are no completely trusted users. Only the TCB is completely trusted.

1. TCB- completely trusted
2. System Administrators - Trusted to administer parts of system when working together, but not as individuals
3. System Auditor - Trusted with knowledge of all actions on system, but not to affect policy decisions.
4. Compartment A users - Users trusted with their own information up to compartmented A
5. Compartment B users - Users trusted with their own information up to compartmented B
6. Secret users - Users trusted with their own information up to secret
7. Uncleared Users - trusted with their own information
8. Outside World - hostile countries, etc.

This system can be envisioned with a great many constraints on the actions of users. The two person administrative control and the controls on the auditor are examples. It is also fair to say in this example that all the application code on the system (i.e. everything a user can execute) has some level of trust in an effort to prevent trojan horses. Guidelines for the introduction of software including analysis of the software will permit the policies, including DAC and object reuse to be strictly enforced.

Building Systems

If systems were custom built and designed specifically for the security policy of the purchaser very little more would need to be said. Unfortunately, most systems today are built by a specific site to meet their needs from collections of existing products from commercial manufacturers. This means that each site must attempt to construct a system that is secure as a whole out of pieces that may have more or fewer features and more or less assurance of correct implementation.

The two problems that then present themselves are partitioning the security policy among the components and defining the TCB boundary both within each component and as an entity once the components are combined. An accurate and detailed interface security dependency description embedded within a view of the system architecture as presented by an NSAD can be a large step forward in providing information to compose the components.

As was stated earlier, developers need to improve the documentation of the security dependencies of and policies enforced by their products. Integrators need to relate their experience to developers to guide the creation of products with easily composable and useful policies. Certainly with communication components can be developed for at least some broad classes of policies although specialized policies implemented in only a few systems still present unique problems for composibility.

For today, however, the poor development documentation leaves the integrator and evaluator with a significant amount of work even for widely used policies. In the absence of NSAD-like documentation, a security policy and interface assumptions need to be developed for each component. These can then be compared to determine if the policy seems to be coherently implemented by the combined system.

Evaluating Security

TPEP evaluations seem to have begun to look at the world more in the manner advocated by this paper. Obviously, within the constraints of looking at systems for conformance to general requirements rather than applicability to a particular environment, there is no way to look at the complete range of possible interface policies. However, some characterizations can be made even for general policies.

Several recently evaluated products have been general purpose systems with a network interface evaluated against the TNI. Those systems provided the evaluation team a NSAD describing the security assumptions at the network interface. The product NSADs will hopefully prove useful to integrators and accreditors when combining products evaluated with these documents. In addition, more effort is being put into describing the software interfaces for trusted applications both in systems that support applications and in the applications themselves. These examples should provide some experience with creating and using NSADs for combining products that will improve the usefulness of these documents over time.

Unfortunately, the limited scope of general purpose evaluations means that even widely used trusted application software for a particular product may not be in the TPEP evaluated TCB of that product. Integrators and acceditors are then forced to examine the trusted applications for each new composite system. This is particularly unfortunate if the applications enforce additional policy which the ultimate system security policy requires but the TCSEC did not.

References

- [1] Anderson, J.P. *Computer Security Technology Planning Study*, ESD-TR-73-51, vol. I, AD-758 206, ESD/AFSC, Hanscom AFB, Bedford, MA, October 1972.
- [2] DoD 5200.28-STD *Department of Defense Trusted Computer System Evaluation Criteria*, December 1985.

- [3] Hosmer, Hilary H., "The Multipolicy Paradigm," Proceedings of the 15th National Computer Security Conference, October 1992, Baltimore, MD, pp. 409-422.
- [4] NCSC-TG-005, *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, Version 1, July 1987.
- [5] NCSC-TG-021, *Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria*, Version 1, April 1991.

THE COMPOSITION PROBLEM: AN ANALYSIS

Guy King
Computer Sciences Corporation
7471 Candlewood Road
Hanover, MD 21076
410/684-3573

"Intellectuals seek only to *understand* the world; but the point is to *change* it." (paraphrase of Karl Marx)

Abstract. Because the "cash value" of previous construals of the so-called composition problem has disappointed expectation, this paper reexamines the foundations of this problem. It discusses the grammar of secure system composition and provides a conceptual analysis of the term "composable"; records insights suggested by the juxtaposition of secure automated information systems with nonsecure systems and with trusted products; touches on the "logic" of secure system composition; and offers a new statement of the composition problem.

Keywords. Composition Problem, Secure System Integration, Standards, Trusted Products

1. Introduction.¹

In the current economic environment, government agencies and commercial enterprises desire as much as possible to satisfy their automated information system (AIS) needs through the use of commercial off-the-shelf (COTS) products. For secure AISs this has proved so difficult that the difficulty has been given a name: the "composition problem."

The composition problem has been defined in various ways. One common definition is along the following (epistemic) lines:

The composition problem is the problem of knowing what inferences one can make about the security properties of a composed system from the security properties of its constituent products.

The constituent products of a secure system are ideally trusted COTS products which, having been successfully evaluated against a set of security requirements by the National Computer Security Center (NCSC), are named on the NCSC's Evaluated Products List (EPL).

Since great quantities of ink have been spent to solve this problem and still the problem remains, a fresh look at the foundations of the problem seems in order. This paper documents an attempt at such a look.

2. The Grammar of System Composition.

'Combining two products to form a system' is logically prior to 'combining two trusted products to form a secure system'. Put another way: if two trusted products cannot be combined to form a system, they certainly cannot be combined to form a secure system. So before analyzing the grammar of secure system composition, we shall examine the grammar of system composition.

Consider the utterance

"Products X and Y are composable into a system S."

¹This paper is based on work performed under Contract MDA904-93-C-B053 for the National Security Agency (NSA).

Syntactically, "composable" is a polyadic operator: it relates multiple constituents with some whole, thus:

C ([X,Y],S)

In the semantics of everyday English, "composable" means "able to be combined." In the utterance "Products X and Y are composable into a system S," we do mean that the two products are able to be combined; but we mean more than this.

We mean also that they are able to be combined together to form some sort of unity, namely, the system S. When combined, the products do not merely exist "side-by-side," as it were, without any interrelation. Together they form a system, as we say, and together they satisfy the system's security requirements.

When we say that two products are composable, we also mean to imply something about the ease of combining them. We do not mean that their combination is merely logically possible. We mean that it is relatively easy to combine the products to form a system. We mean that we should neither have to change the products nor to develop additional code in order to be able to combine them.

3. Systems and Products.

It is remarkable that the composition problem is viewed as pertaining to the composition of *secure* AISs from *trusted* (and *untrusted*) products but not to the composition of other AISs from *untrusted* COTS products. (Let us call these other systems "nonsecure," meaning by this that security requirements do not figure importantly in their design.)

This is remarkable because secure AISs are, surely, just a species of AIS. It is a mystery, therefore, why composition should be a problem for the former but not the latter.

This apparent anomaly is an important clue to the solution of the composition problem.

Reflection reveals, first of all, that integrators of nonsecure systems from untrusted products have also faced the composition problem. They, too, have experienced difficulties trying to make two AISs interoperate, and trying to port an application to a different operating system (or a new release of the old one). They speak of "the interoperability problem" or "the applications portability problem" rather than of "the composition problem"; but "What's in a name? That which we call a rose/By any other name would smell as sweet" (Shakespeare, [8])--and "the composition problem" as fetid.

Reflection indicates, second, that, though the perception is false that integrators of nonsecure systems have not faced the composition problem, there is good reason for this perception: an increasingly large number of untrusted COTS products are composable into systems. For example, popular COTS wordprocessing, spreadsheet, graphics, and DBMS software products are portable to a number of widely used operating systems; and various mainframe computers and workstations can interoperate through the use of standard communications protocols (TCP/IP, etc.).

One strand of the history of the integration of nonsecure systems from untrusted products is, then, the story of the progressive conquest of the composition problem.

Study of that history reveals, thirdly and most importantly, the means by which, in the sphere of nonsecure systems, the composition problem is being solved. It is being solved through the implementation of **standardized interfaces, protocols, and data formats.**

This lesson is embodied in NIST's *Applicability Portability Profile: The U.S. Government's Open System Environment Profile* [6] (hereafter, "APP Guide"). Using

IEEE's POSIX Working Group P1003.0's OSE Reference Model as a basis, the Guide defines seven service areas (Operating System [OS], Network, Data Management, Human/Computer Interface, etc.), each of which "addresses specific components around which interface, data format, or protocol specifications [i.e., standards] have been or will be defined" ([6], p. 10).

As a glance at the APP Guide [6] or its Department of Defense (DoD) analog, the *Technical Reference Model and Standards Profile Summary* [1] (hereafter, "TRM&SPS"), shows, there exist very few standards in the area of security. And of the few which do exist, fewer still are rated high enough (using the APP Guide's rating factors of level of consensus, product availability, completeness, maturity, stability, de facto usage, and problems/limitations) to be included in the Profile.

The TRM&SPS [1] names the *Trusted Computer System Evaluation Criteria* [3] (hereafter, "TCSEC"), as one of the security standards. The TCSEC, of course, is the document containing the sets of requirements against which candidate products are evaluated by the NCSC for trustedness. It is by now notorious that the TCSEC, published in December, 1985, was written with standalone mainframe computers as the paradigm of a trusted product; since its existence predates the rise of networked hosts and workstations, its requirements do not address interfaces or protocols for the communication of security information. (Security information includes the following: userids, passwords, audit data, groupids, security labels, privileges, authorizations.) This, quite simply, is the reason the composition problem is so severe for trusted products: since composability is not a TCSEC requirement, trusted products are not designed to be, hence are not, composable.

This explanation of the problem of composition for secure systems arose from a comparison of secure with nonsecure systems. It could have arisen just as well by comparing secure systems with trusted products.

Trusted products and secure systems (or, for that matter, products and systems generally) do not in concept differ in kind but exist on the same continuum, as it were. We say that trusted products are components of secure systems; but a trusted product can itself be a system. Also, a secure system can be composed of other secure systems. And a trusted product can consist of trusted products (for example, of an evaluated hardware platform and of an evaluated operating system). But we could as well call this "compound trusted product" a secure system, since its components are trusted products.

Given that the terms "trusted product" and "secure system" are thus relative, and that they can be applied to the same referent, how is it that INFOSEC professionals speak of the composition problem with respect to secure systems but not with respect to trusted products? Expressed otherwise: why is it possible (relatively) easily to engineer trusted products but not secure systems?

The answer is obvious: a trusted product is relatively easy to engineer because the interfaces, protocols, and data formats necessary to compose the product's modules are defined by the product's developers. If in the development of a trusted product no developer cooperated with any other to define the interfaces, protocols, and data formats needed for inter-module communications but each made up his/her own definitions, then the composition problem would exist for trusted products, too.

The composition of secure systems is a problem because the security-relevant interfaces, protocols, and data formats have been defined in no standards; the result is that each trusted product developer has had no choice but to define the external security interfaces, protocols, and data formats by his/her own lights. The Babel thus produced is the problem of composition.

4. The Grammar of Secure System Composition.

When we say that two products can be combined to form a secure AIS, we mean that in the resultant combination there is minimal loss of the security properties of the constituent products; or, a maximal subset of the security properties of both products is inherited by the composed system.

We mean further that this maximal inheritance, this minimal loss, proceeds in a rulelike manner. If it did not, then when users had need of a system with particular security features and assurances, they would have no basis for selecting these constituent products rather than those. If security property inheritance were not rulelike but arbitrary, then to compose, say, a B1 system, one would have no more reason to select as constituent products ones satisfying B1 requirements than ones satisfying B2 requirements, or C2, etc. (One could as well speak here of the CSn or LPn Protection Profiles of the *Federal Criteria* [7] as of TCSEC evaluation classes; or of the functionality/assurance ratings of international standards. TCSEC ratings are instanced here because of their widespread familiarity.)

It was and is the intent underlying the EPL that users be able to infer, from the need for an AIS with certain security properties, the security properties of the trusted products from which to compose the system. This has also been the actual practice within DoD for years: if, for example, according to DoD Directive 5200.28 [2], users need a system which satisfies TCSEC B1 requirements, they are to try to satisfy this need by composing their system of COTS operating systems, DBMSs, etc., which have an EPL rating of at least B1.

The rules of security property inheritance constitute the "logic," then, which secure system composition from COTS products *should* obey. Examples of theorems in this "logic" are:

- * A system composed of a trusted and untrusted component (where the untrusted component enforces none of the system's security requirements, and can circumvent none) has the rating of the trusted component.
- * A system composed of (say) two B1 components is itself B1.
- * A system composed of (say) B1 and C2 components has an overall digraph equal to the lower of the components' digraphs--in this case, C2

(See also the NCSC *Trusted Network Interpretation of the [TCSEC]* [5], Section A.2, "Composition Rules.")

When trusted products are designed to satisfy such theorems, then users will have the basis needed for selecting from among them when aiming at a system of a certain security functionality and assurance (B1, say).

To summarize what is meant by 'two [or n] trusted products are composable into a secure AIS':

The two products can be easily integrated with one another, in true "building block" fashion,

- (1) without either of them being modified,
- (2) without the use of integrators' "glue" (i.e., unevaluated trusted code), and
- (3) with maximal inheritance by the system of the products' security properties, according to a set of specified rules.

The results of this attempt to look again at the foundations of the composition problem suggest that we ought to reconstrue it along the following lines:

- (1) To identify those properties which trusted products must come to have if they are to be composable in the above sense; and
- (2) To draft and promulgate standards which require such properties of trusted products.

5. Conclusion.

In Section 1 above was described an epistemic construal of the composition problem; and the remaining sections ended with a reconstrual of this problem. In this concluding section an analogy will be drawn which, it is hoped, will confirm the error of the epistemic version and the correctness of the reconstrual.

Imagine a world in which bricklayers seek to construct stable buildings using bricks **not** of the sort currently in use (rectangular, about 2.25" by 3.25" by 8", and of moist clay hardened by heat) but of varying sorts--some rectangular, some square, some pentagonal, some spherical, etc., with differing centers of gravity; some 2" by 3" by 7", some 5" by 6" by 13", etc.; some made of hardened clay, some of steel, some of plastic, some of papier-mache, etc. Imagine next that, in response to the great difficulties they encounter in achieving stable houses, these bricklayers devote significant effort to cataloging the stability-relevant properties of the various sorts of brick as well as the kinds of relations which the sorts can bear to one another, and then to analyzing this data to discover what inferences they can draw about the stability-relevant properties of the resulting houses from the stability-relevant properties of the constituent bricks.

The "construction problem" which the bricklayers expended so much effort to solve does not differ in essentials from the epistemic version of the composition problem. Guided by this statement of the problem, INFOSEC professionals have sought answers to analogs of the bricklayers' questions:

- (1) What kinds of relations can trusted products bear to one another in systems?
- (2) What are the security-relevant properties of trusted products when in such relations?
- (3) What inferences can be drawn about the security-relevant properties of the composed system from the security-relevant properties of the constituent products?

As with the bricklayers, so here: despite such cataloging and analysis, composing secure systems from trusted products is **not significantly easier**. Secure system integrators continue to bemoan the difficulty of their task; and accounts of their experiences remain a staple of INFOSEC Conferences.

Nor is this surprising. The epistemic version of the composition problem is shown to be mistaken by the Fallacy of Composition, which has the following form:

- (P) The members of a set, or the parts of a whole, have some property P; therefore,
- (C) the set, or the whole, has property P.

One example of this fallacy is:

all the parts of this machine are lightweight; therefore,
this machine is lightweight.

Another example is:

all the components of this AIS are secure; therefore,
this AIS is secure.

In short, it is *fallacious* to argue from the properties of components to the properties of the whole.

NCSC C Technical Report 32-92, "The Design and Evaluation of INFOSEC Systems: The Computer Security Contribution to the Composition Discussion" [4], expresses it this way (p. 12):

"...in general, one cannot ask about the composition of an arbitrary set of elements--one does not interconnect a VCR, a carburetor, and a toaster, independently of any system context, and then ask, 'what policy is enforced?' Composition does not occur by accident!"

It's true: a secure system composed of trusted products must be *engineered*.

This entails, among other things, that the component products must be designed to be securely composable with one another. That these products are not currently so designed is shown by the difficulty encountered by all who attempt to compose a secure system from them. Q.E.D.: current trusted products lack some properties necessary for them to be composable into secure systems; and the composition problem ought to be restated to address this lack.

The bricklayers' analysis was ineffective because it **did not change the properties** of the various types of bricks used; bricks continued to lack the (standardized) properties necessary to make stable construction easy. Once appropriate standards were articulated and enforced for the production of bricks, easy construction of stable houses quickly followed.

Likewise, current trusted products lack some properties necessary for them to be composable into secure systems. Once standards specifying those properties are articulated and enforced, easy composition will follow.

Here, as so often in life, the trick is to know, of the things which are changeable, what ought to be conserved and what changed. Trusted products are changeable, and far more and better security can be realized if they are required to accord with appropriate security interface, protocol, and data format standards.

REFERENCES

1. DISA, *Department of Defense Technical Architecture Framework for Information Management (TAFIM)*, Vol. 2, *Technical Reference Model and Standards Profile Summary*, Version 2.0, Final Draft, 1 Nov 93.
2. DoD Directive 5200.28, "Security Requirements for Automated Information Systems (AISs)," 21 Mar 88.
3. DoD 5200.28-STD, *Trusted Computer System Evaluation Criteria (TCSEC)*, Dec 85.
4. NCSC C Technical Report 32-92, "The Design and Evaluation of INFOSEC Systems: The Computer Security Contribution to the Composition Discussion."
5. NCSC-TG-005, *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria (TNI)*, Version 1, 31 Jul 87.
6. NIST, *Application Portability Profile (APP): The U.S. Government's Open System Environment Profile*, Version 2.0, May 93.
7. NIST & NSA, *Federal Criteria for Information Technology Security*, Vols. 1-2, NIST & NSA, Version 1.0, Dec 92.
8. Shakespeare, *Romeo and Juliet*.

"MAKING DO WITH WHAT YOU'VE GOT"

Janis W. Berryman
The Boeing Company
P. O. Box 3707, MS 8E-74
Seattle, WA 98124-2207

Bruce F. Kennedy
Cubic Applications, Inc.
PSC 303; Box 50
APO 96204-0050

ABSTRACT

"Making Do With What You've Got" is a common-sense approach to enhancing system security with low-cost COTS (Commercial-off-the-Shelf) hardware and software. Today's economy and reality of industrial espionage is forcing companies to look at the cost of security and make choices. The choice to have a high level of security can be accomplished by purchasing National Computer Security Center (NCSC) approved software, i.e., A1 level. Most systems today are operating at either an NCSC-approved or designed-to-meet C2 level of trust. This paper offers an alternative to companies faced with the dilemma of needing additional security beyond C2, but having too few dollars to provide the added protection of approved A-1 rated software. The solution is "Making Do With What You've Got" which implements, in a more rigorous fashion, selected features of NCSC Division B by addressing specific elements of the B1/B2 protection. The approaches taken by these authors for processing classified information are not to be interpreted as an elimination of the use of NCSC trusted systems. In fact, the NCSC levels are a reference model for achieving a cost-effective solution. This paper includes a case study using Digital Equipment Corporation VAX/VMS systems to illustrate some practical applications of this concept.

KEYWORDS

Common Sense Approach; Risk Management; NCSC Levels of Trust; Cost-effective Security; VAX VMS; Risk-Assessed Compartmentation.

INTRODUCTION

Today's economy is currently at work in your computer rooms. Disguised as budget crunches and "No More in '94" business plans, it has forced computing security professionals and system administrators to become innovative, maybe even daring, if his/her company is to survive The Nineties.

Historically, government agencies employed the concept of risk avoidance to ensure a secure system or network. By definition, the more money you spent, the more secure your system. Survivors of this way of doing business can espouse to the wounds of users and management demanding more access to a system or network, while customers were levying more security requirements - making the needed access almost impossible to achieve.

Rather than continue with risk avoidance, companies must begin anew with risk management. More often than not, companies have a requirement to ensure separation of data and individual accountability. One way this could occur is with a requirement to protect sensitive or proprietary information in the commercial or unclassified world. Examples would include separating Human Resource or Medical data from engineering, design or technical data. Perhaps more familiar is the requirement to separate data based on a need-to-know or the need for individual accountability. Either or both of these requirements are found in classified environments. When this occurs, users are required to be uniquely identified and the operating system must maintain an automated audit trail. To do this, system administrators and computing security professionals rely on systems which have been rated at the NCSC Class C2 level. Class C2 systems are chosen because these systems are at the lowest hierarchical point of trusted systems designed to provide individual accountability and need-to-know protection. A C2 level of trust only provides discretionary access control which restricts access to objects based on who the subject is and/or the group to which the subject belongs. A C2 level of trust cannot provide the capability to put trusted labels on the output. A human must review the content in order to provide the correct labeling of the output.

In general most companies with two or more projects requiring a protection of need-to-know and individual accountability have basically the following ways to do business: (1) dedicated resources (duplicate equipment); (2) dedicated periods, where each project schedules usage of the same resources; or (3) compartmented, where both projects are processed simultaneously within a trusted computer system which is relied upon to maintain the project and data separation.

Processing with dedicated resources provides a company with the possible advantage of having an in-house system that may suffice. The disadvantage of dedicated resources is higher costs and longer lead times for facilities and hardware/software procurements.

Dedicated periods processing implies that a company has the advantage of having existing resources to support both projects. The disadvantages of dedicated periods of processing are the need to schedule project time and inefficient computer use.

Traditionally, companies faced with this dilemma have chosen NCSC-rated equipment for a trusted computer baseline (TCB). Defense companies relying on the TCB have the advantage that it is an easier way of getting customer approval to process. Disadvantages of processing in the compartmented mode in the past have been the cost and schedule impacts of procurement and implementation; sometimes loss of compatibility with other machines; cost of any required conversions when changing platforms to obtain and maintain a TCB; and often times an increase in operating costs due to the unique setup.

Although viable options for some companies, the above three solutions all have one security concept in common - risk avoidance at whatever cost the company is willing or required to expend.

RISK MANAGEMENT

The need for companies to be innovative, cost effective, and results oriented is causing some companies to look at a fourth option; a new security concept called risk management. Risk management is a logical, rational, and cost effective approach that can meet the intent of the NCSC, B1 level of trust. Risk management meets the intent by combining physical, personnel, and system access controls with dedicated system and security administration providing rigorous configuration management. Sufficient controls are put in place to minimize human error and to provide audibility and accountability for user actions and system activity. Applied correctly, risk management provides system and security administrators detailed knowledge of the circumstances under which users may gain unauthorized access to specified data, and enables them to implement additional countermeasures, if needed. Risk management lets the company be in control of setting its priorities. It ensures that all projects and users have state-of-the-art technology available for their use, and provides the tools for a company to work smarter and to enhance competitiveness.

The accepted probabilistic equation for risk is threat times vulnerability times consequence less countermeasures. Threat is the probability that something will attack your system. Vulnerability is the probability that if an individual targets your system for an attack, he/she will be successful. Consequence is the damage done when a successful attack occurs. Countermeasures are any provision put in place to minimize the vulnerabilities of your system.

Rather than use the probabilistic model, risk management evaluates all elements of the risk definition, understands the cost of a successful attack either in terms of dollars or national security, and judiciously applies countermeasures. In short, risk management is understanding what you are willing to pay for a loss - either up front or after the fact.

Figure 1 is a process flow showing the steps a company would take prior to implementing a risk-assessed security model.

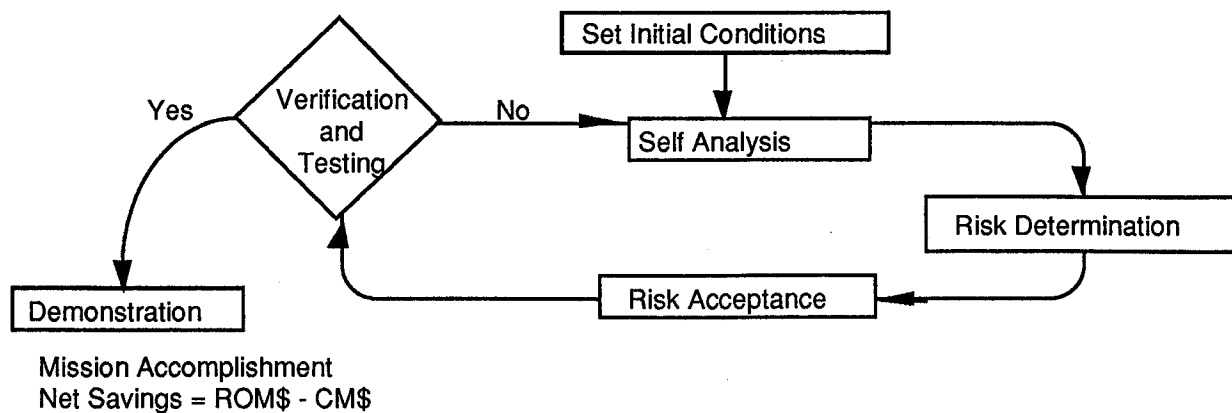


Figure 1 - Process flow for implementing risk-assessed security model

Once a company has reviewed its requirements and determined that a risk management approach is appropriate, it must then *set the initial conditions*. Setting the initial conditions is an examination of the options available and a determination of the rough order of magnitude (ROM) of costs for each option. Available options may be somewhat restricted, especially in the defense industry where accrediting authorities determine criteria for viable alternatives. Less restrictive in the commercial world, a company may be expanding its business base by acquiring another business, or reducing its operations through consolidation or centralization.

To adopt a risk management security profile, the company must conduct a *self-analysis* of its existing security profile and assess the cost of maintaining that profile. Remember that a company's total security profile includes physical, personnel and system access(es). A company should ensure good physical access controls over computer rooms, network trunks and placement of terminals and printers. Likewise, a company policy defining users who are authorized to have computer accounts should be in effect. During the self-analysis, a company must study and understand the level of security needed. Items such as proprietary information, the Privacy Act, regulatory agencies demands, and management philosophy are just a few of the areas that will need examination.

Risk determination requires an assessment of threat, vulnerabilities, countermeasures, and cost of a loss. Table 1 on the following page is a matrix showing risk determination factors.

Risk acceptance occurs when a company understands what data is being protected; how much it costs to protect that data; what data needs to be protected in the future; how much the company is willing to pay for future protection; and chooses an option available. A company determines the degree of risk it is willing to take by: (1) understanding the security features inherent in the hardware and software used; (2) knowing what system administration buys; and, (3) implementing and maintaining configuration control over the hardware and software to ensure the accuracy of the risk determination matrix.

Inherent security features may not provide any security unless the capabilities of that system are properly evaluated and some effort is expended to implement them properly. A good example is the VMS 5.5 operating system. When the VMS operating system was first evaluated by the NCSC, Version 4.3 was rated as meeting a Division C, C2 level of trust in 1986. By 1994, Digital Equipment Corporation (Digital) has expended a lot of effort during its Rating Maintenance Phase (RAMP) in bringing this operating system forward, and has not overlooked some of the security controls required by commercial users of its product. As a result, VMS version 5.5 has are significant controls that a knowledgeable system administrator could take advantage of to

Table 1 - Risk Determination Matrix

DESCRIPTION/NCSC REQUIREMENT	THREAT	VULNERABILITY	COUNTERMEASURE	RISK
3.1.1.3 LABELS	Individual workaround requiring concentrated effort	Network copy using explicit access via batch files.	Disable explicit copy by using secondary passwords; audit trails; no interactive access to compartmented host.	Low
3.1.1.3.2.3 LABELING PRINTED OUTPUT *AND*	Accidental spillage; markings are not mandatory.	An authorized user sees unauthorized data.	Printing under control of centralized computing facility staff; all print jobs require use of banner pages for front and back of printouts.	Low
3.1.2.2 AUDIT ANY OVERRIDE OF PRINTED OUTPUT MARKINGS	DITTO	DITTO	DITTO	Low
3.1.1.4 MANDATORY ACCESS CONTROLS	Individual work-around of device ACLs requiring concentrated effort. Activity unable to be monitored.	An operator may modify configuration either deliberately or inadvertently.	Strict configuration management to ensure configuration integrity; audit trail documentation.	Low
3.1.2.2 AUDIT	Unable to adequately monitor system.	Lack of knowledge on system activity.	Use knowledgeable system and security administrators to review system activity.	Medium
3.2.2.1.1 TRUSTED COMMUNICATIONS PATH	Ethernet monitor. Information intended for another host may be monitored.	An authorized user sees unauthorized data.	Ethernet is fully enclosed within limited-access computing facility.	Low
	Password grabber program emulating login.	Legitimate user USERIDs and passwords may be compromised.	Login terminals use VMS secure server feature (break key).	Low

Note: The Risk Determination Matrix is used as a tool to identify those areas covered by the B1 NCSC requirements which are not initially met with a C2 VAX/VMS system. Each requirement is evaluated for associated vulnerability and threat. Countermeasures are developed to offset the vulnerability and a risk determination is made which looks at the likelihood of the threat and the potential of the vulnerability given the countermeasures. This table shows those risks being assessed as Low, Medium, or High, but any scale could be used. For this Risk Determination Matrix, a risk of High was considered unacceptable.

upgrade a security profile to meet the intent of a level higher than C2, without expending the dollars to buy such a system or additional third-party products.

The second important item to risk management is good system administration. An infrastructure of well-trained, knowledgeable individuals who understand the system and are capable of implementing the system's available tools; who can administer it properly; and, who regularly maintain the established system and security administration baseline. Proper administration requires all security controls to be explicitly addressed. Users are allowed just enough access to do their job efficiently, and are encouraged to exercise discretion in permitting access to files. Privileges are granted only if absolutely necessary - not by default. In addition to user and system access, the company's computing security staff should be sufficiently trained to make effective use of their own system accounts to monitor system and audit trail activity. This would also allow for an informal implementation of the Trusted Facility Management requirement found in Division B, Class B2.

Regular maintenance is important to catch potential problems early. It will ensure that users are not using "workarounds" for security. Early detection of security problems lets users know there is proactive administration, as opposed to reactive crisis-management of security incidents.

Implementation of a sound configuration management program is a required baseline for maintaining a level of security. There is nothing wrong with a company knowing what their assets are, where they are located, how they are being used, and how they can be used in the future. Without a constant level of security, you cannot achieve a higher level.

In a risk management approach, the resulting security profile requires achieving configuration control of your system administration as the baseline. Normally, system administrators are familiar with only software control, i.e., application, versions, licenses, compatibilities. Under risk management, the operating system parameters will be maintained similar to a software development environment. In a software development environment, lines of source code, version configurations, test cases, bug fixes, etc., are under configuration control. In a risk managed security environment, configuration management procedures will be in place encompassing items like: system administration parameters; default access control lists (ACLs); command procedures.

After a company has determined its risk acceptance position, *verification and testing* is performed. The verification process has the following steps: (1) each countermeasure (see Table 1) is reviewed; (2) a course of action determined; (3) appropriate implementation is in place; and (4) effectiveness of each countermeasure is tested. For example: Using application of mandatory access controls requires placing Access Control List(s) (ACLs) on each device. The countermeasure is implemented when the ACLs are in place. The effectiveness is tested by a three step demonstration as shown in Example 1, beginning with Step 1, Category B device ACL:

Example 1, Step 1, Category B device ACL:

```
VAX1 > show device/full CATB_DISK
```

```
Disk VAX1$DUB101:, device type RA70, is online, mounted, file-oriented  
device, shareable, available to cluster, error logging is enabled.
```

```
·  
·  
·
```

```
Volume status: subject to mount verification, file high-water marking,  
write-through caching enabled.
```

```
Device access control list:
```

```
(IDENTIFIER=[CATB,*],OPTIONS=PROTECTED,ACCESS=READ+WRITE+EXECUTE+DELETE)  
(IDENTIFIER=[*,*],OPTIONS=PROTECTED,ACCESS=NONE)
```

Example 1, Step 2, ACL for Category B device top-level directory (Master File Directory or MFD):

```
VAX1 > directory/security CATB_DISK:[000000]000000.dir
Directory CATB_DISK:[000000]
000000.DIR;1          [STAFF,SYSTEM]          (RWED,RWED,RE,E)
                   (IDENTIFIER=[CATB,*],ACCESS=READ+EXECUTE)
                   (IDENTIFIER=[*,*],ACCESS=NONE)
```

Example 1, Step 3, Category A user attempting to access the Category B device:

```
$ SET DEFAULT CATB_DISK:[000000]
$ DIR
%DIRECT-E-OPENIN, error opening CATB_DISK:[000000]*.*;* as input
-RMS-E-PRV, insufficient privilege or file protection violation
```

An entry is then made into audit log by the operating system. If the testing is unsuccessful, you would return to the self-analysis step.

Throughout the whole process, a company must maintain thorough written documentation for each step. *Demonstration* requires affirmation that the process flow was completed in total. After a successful demonstration, an analysis of the cost savings would be conducted. At this point, defense industries would begin the process to get approval from their accrediting authority.

The formula for *cost savings* is: Net Savings = ROM \$ - countermeasure \$. In the beginning, when initial conditions were set, options were identified and a rough order of magnitude of costs were associated with each one. With the information derived from completing the process flow and using the ROM and countermeasure dollars in the cost savings formula, a company can realize its best option.

CASE STUDY

The risk-assessed compartmented system methodology was applied to a small computing lab with three large-scale VAX platforms and one microVAX all running VAX/VMS V5.5-2. One VAX is a shared resource in batch mode for users on the other VAXes and the microVAX. The main purpose of implementing a risk-assessed compartmented system in this environment is to provide the same level of computing resources for two projects (Category A and Category B). Category B requires only a small amount of interactive processing and can utilize a microVAX for that purpose. However, it requires computing resources for batch jobs which far outstrip available funding.

Applying the process flow, identified above, demonstrated that a significant cost savings would be obtained by sharing the large Category A batch machine (VAX3). Thus, the challenge is to implement acceptable controls in this environment which can be accepted for a risk-assessed compartmented system.

There are three major areas addressed by the implementation for a specific system configuration on VMS: 1) separate characteristics for each account to identify clearance and authorization for access to a specific category of data; 2) the use of mandatory access controls through device Access Control Lists (ACLs) and login command procedure elements; and, 3) establishment of baseline network controls to restrict communications to authorized nodes and usernames.

A **basic configuration** would consist of a microVAX (mVAX) dedicated to run Category B only. VAX2 is set up to run both Category A and Category B batch jobs with each project having its own data disk(s). The diagram shows that the mVAX Category B users are allowed access

only to the VAX2 Category B data. MVAX users are not able to access VAX2 Category A data. Mandatory access controls (device ACLs) are in place to prohibit Category B users from accessing Category A data. Figure 2 shows the DECnet configuration of the MicroVax to the VAX and available node access.

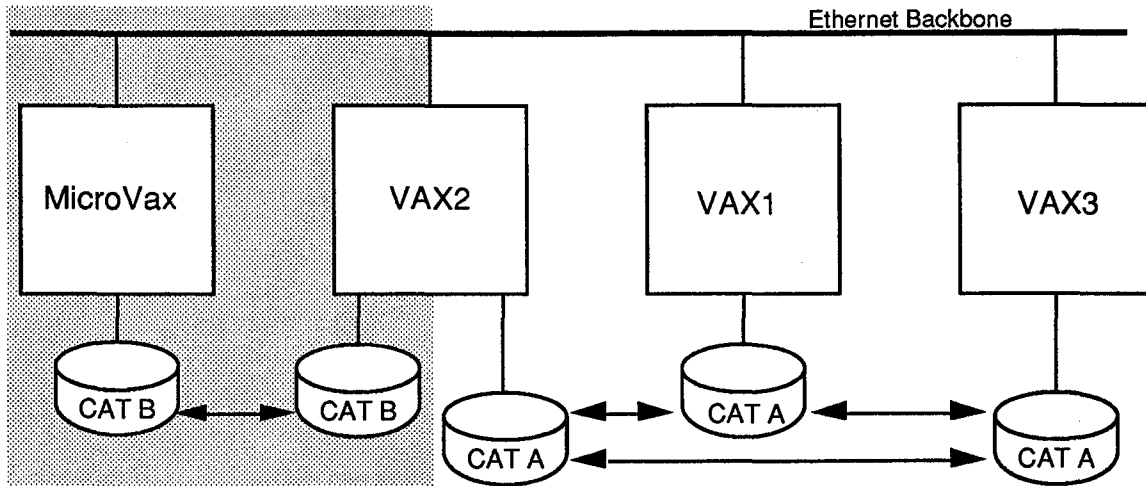


Figure 2 - VAX DECnet configuration

Both VAX1 and VAX3 are Category A systems. The diagram shows that they have access to each other. VAX1 users are only able to access (if authorized) VAX2 Category A data. Protections are set to prohibit Category A users from accessing VAX2 Category B data. DECnet is configured so that dedicated Category A machines (VAX1 & VAX3) cannot communicate with the dedicated Category B machine (mVAX). The NCP (Network Control Program) files that configure and control DECnet have file protections in place to prohibit users from accessing or altering them.

During *account setup*, users for specified categories will have separate accounts assigned with unique group User Identifier Code (UIC) numbers (i.e., not the same as a group using another category) and an associated rights identifier for that category and separate account charge. These accounts must be limited to TMPMBX and NETMBX privileges.

Use of Mail is not affected by device ACLs, UICs, or Rights Identifiers. In order to disable Mail, the DISMAIL flag must be set in the user's UAF record.

Mandatory Access Controls would require the use of Separate disks for Category A and B users. These disks should be restricted with the use of a device Access Control List (ACL). This ACL would limit access on that disk to the specified category rights identifier only. Example 2 shows one possible configuration for a device ACL using the VMS ACL editor. The combination of the two ACL's is necessary to deny access to all other users. Also, default protections on the device (S,O,G,W) need to be modified to limit access. As long as a user matches the group and has GRPPRV or anyone who has SYSPRV, he/she can circumvent the ACL, unless default protection is set to (S:,O:,G:,W:)--no access for anyone outside of the ACL. This may require that the ACL include access for SYSTEM to do backups, or else ensure any process doing backups has the BYPASS privilege.

Example 2, Device ACL example using VMS ACL editor:

```
$SET ACL/EDIT/OBJECT_TYPE=DEVICE DUB100                ! Omit colon

(once in ACL editor, add the following ACE's)

(IDENTIFIER=[CATB,*],OPTIONS=HIDDEN+PROTECTED,
ACCESS=READ+WRITE+EXECUTE+DELETE)
(IDENTIFIER=[*,*],OPTIONS=HIDDEN+PROTECTED,ACCESS=NONE)

[ctrl-z] to exit

$SHOW ACL/OBJECT_TYPE=DEVICE DUB100                    ! Omit colon
```

Device ACLs are not saved after system shutdown, and must be restored either manually or by a command procedure. The **HIDDEN** option can be used if users should not know about the implementation of the security controls. However, it makes setup more cumbersome as it can only be added through the ACL editor. ACLs on the Master File Directory (000000.DIR) for each disk are retained after system shutdown and could be used to backup device ACLs. Queue-based ACLs would backup printer device ACLs.

Note: Protected and hidden ACLs operate under special rules for modification and deletion. Any testing done with the ACLs should be done without these options set. After testing is complete, the ACL editor should be used to add those options. The ACLs should be hidden and protected prior to users having access to the system.

Network access controls for DECnet must be implemented to prevent a user from accessing a Category A account from a Category B account and vice-versa.

DECnet access explicitly using a username and password can be denied by requiring all users on the system to have a secondary password. A simple secondary password (such as 'CATB') would be sufficient to deny network authentication through explicit access. A secondary password can be set up on an account by doing *MODIFY/PASSWORD=("",secondpwd) USER* in **AUTHORIZE**.

Using a simple secondary password effectively denies network authentication through explicit access. In the case study, this method should be required for the Category B users. If the user changes the secondary password, then it must be 6 characters long and unique in the same sense as the primary. This can be bypassed by allowing the user to log on once with a single password and change it. Example 3 shows how the account can be modified.

Example 3, Initial Account Setup in Authorize

```
$RUN SYS$SYSTEM:AUTHORIZE
UAF> COPY/OWNER="John Q. Public"/UIC=[427,77]/DIRECTORY=[JQP]-
      /NOPASSWORD          DEFCATB      JQPCATB
UAF> MODIFY/PASSWORD=FIRSTPWD          JQPCATB

<User logs in to account and selects initial password>

$RUN SYS$SYSTEM:AUTHORIZE
UAF> MODIFY/PASSWORD=("",CATB)          JQPCATB
UAF> MODIFY/NOPWDEXPIRED                JQPCATB
```

This will give all Category B users a short, simple secondary password and disable explicit access over the network.

Proxies have been set up on the mVAX only for VAX2. VAX2 proxies to mVAX are only for designated Category B accounts. No proxies from the mVAX are set to VAX2 Category A accounts.

The use of SET HOST should be disabled. It can be limited or completely disabled. Limiting it can be accomplished through DCL commands in SYSSYLOGIN.

The DECnet configuration in NCP (Network Control Program) needs to be modified to allow access only from authorized hosts. This will ensure that other Category A machines on the network will not have access to the Category B mVAX. Example 4 shows the parameters used to disable connectivity to the mVAX (Step 1) and an example of what error messages are displayed when a user tries to access the mVAX from VAX1, and VAX2 (Step 2).

Example 4, Step 1, NCP parameters to disallow access from a specific node

```
ncp> set node 1.4 access none          ! changes to volatile database
ncp> define node 1.4 access none ! changes to permanent database
```

Commands to remove the node name associated with the node address from the database (if they exist already) are:

```
ncp> clear node 1.4 name              ! changes to volatile database
ncp> purge node 1.4 name              ! changes to permanent database
```

Example 4, Step 2, Attempted access from an unauthorized machine:

```
VAX1 > SET HOST VAXCATB
%SYSTEM-F-NOSUCHNODE, remote node is unknown
```

```
Attempted access from an unauthorized machine using its DECnet address:
VAX1 > SET HOST 1028
%SYSTEM-F-UNREACHABLE, remote node is not currently reachable
```

The mVAX DECnet configuration is set to have VAX1, VAX2 and VAX3 unreachable to users. Node names have also been removed from the DECnet database to disable remote logins.

File protections have been set on the DECnet configuration files and system login files to prohibit tampering. Example 5 shows the files and protections.

Example 5, Critical system files and protections:

```
SYSS$SYSTEM:
NCP.EXE;2          [STAFF, SYSTEM]    (RWE, RWE, RWE, )
NETCONFIG.COM;2   [STAFF, SYSTEM]    (RWED, RWED, RWE, )
NETSERVER.COM;11  [STAFF, SYSTEM]    (RWED, RWED, RE, E)
NETSERVER.EXE;1   [STAFF, SYSTEM]    (RWED, RWED, RWED, RE)
NETPROXY.DAT;1    [STAFF, SYSTEM]    (RWE, RWE, RWE, )

SYSS$SYSROOT: [SYSMGR]
SYLOGIN.COM;94    [STAFF, SYSTEM]    (RWED, RWED, E, E)
```

The *secure server feature* of VMS has to be implemented. This will affect direct-patched terminals only. It is implemented by issuing the DCL command *SET TERMINAL/SECURE-
/PERMANENT <terminal-name>*. This can be included in the SYSS\$STARTUP:STARTUP_V5.COM file.

A *system password* can be required for direct-patch terminal logins. This is implemented through the DCL command *SET TERMINAL/SYSPWD/PERMANENT <terminal-name>*. This could also be included in the SYS\$STARTUP:STARTUP_V5.COM file. To set the system password through AUTHORIZE, *ADD/SYSTEM_PASSWORD=SYSTEMPWD* would do it. It can also be changed by *SET PASSWORD/SYSTEM* at the DCL prompt. Terminals set up with secure server and system password require that the break key be pressed, followed by the system password and carriage return.

These preparations assume that the login source (direct-patch terminals) is defined in advance. Then SYS\$SYLOGIN can be used to verify login from these terminals only. The above terminals are defined in SYS\$SYLOGIN so that users will not be able to access Category B from other areas in the company. All users login initially using the system login file (SYS\$SYLOGIN logical name). This login file tests to see what accounts are Category A or Category B. All Category B accounts are granted access only on the designated terminals and all other accounts are refused.

The above implementation is not meant to reflect a comprehensive configuration, but should show some specific methods for enhancing VAX/VMS to meet a more stringent security requirement. This configuration shows how manually setting up user configuration, rights identifiers and using device ACLs, the intent of Division B, Class B1 security requirements can be met. This configuration is used in the risk determination matrix as the countermeasures to specific vulnerabilities. The approach taken to securing this VAX/VMS network can be applied to other systems as well. Following the risk acceptance process flow and evaluating specific countermeasures for other systems would produce a result which could be analyzed for acceptable risk.

SUMMARY

Staying in business during The Information Age requires companies to bring security issues to the board room and make rational, logical, and cost-effective decisions. These decisions cannot be made without understanding: (1) what you are trying to protect; (2) what your threat and vulnerability state is; and, (3) how much you are willing to pay, either in terms of dollars, time, or effort. The discussion of risk management in this paper was designed to give some insight into the steps required for a company to make the right decision at the right cost.

REFERENCES

Department of Defense Trusted Computer System Evaluation Criteria ('Orange Book'), DOD 5200.28.STD, December 1985.

Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments ('Yellow Book'), CSC-STD-004-85, June 1985.

Guide to VMS System Security, Digital Equipment Corporation, June 1989.

MODERN MULTILEVEL SECURITY (MLS): PRACTICAL APPROACHES FOR INTEGRATION, CERTIFICATION, AND ACCREDITATION

Bill Neugent, Mike Burgoon, Jeanne Firey, and Mindy Rudell

The MITRE Corporation, 7525 Colshire Dr., McLean, VA 22102, U.S.A.

Abstract: This paper summarizes strategies for achieving Multilevel Security (MLS) and describes near-term MLS capabilities being fielded. It outlines approaches for overcoming remaining hurdles in the areas of application integration, certification, and accreditation. Approaches include central registration, automated tools to check fielded capabilities, and common accreditation precedents. Current common accreditation precedents are described that establish new ways of thinking about modes of operation and assurance for MLS capabilities. The paper is derived from and builds upon work performed for the Department of Defense (DoD) MLS Program.

Keywords: Accreditation, Application integration, Assurance, Certification, Computer security, DoD MLS Program, Multilevel security

Introduction*

For years the target of skeptics, multilevel security (MLS) capabilities are now being successfully fielded at an increasing rate. Yet this long-sought success opens the door to new problems. Current MLS fieldings confront hurdles that are different in kind from past MLS efforts, in part because MLS capabilities must be changed frequently in the field and because any change can subtly yet substantially undermine the MLS foundation.

The MLS capabilities currently being fielded are primarily workstations and guards. A major new element complicating *MLS workstation* fieldings is that security is dependent on the effective security integration of frequently changing commercial applications. Security *guards*, on the other hand, support few or no commercial applications but also can change dynamically, because of dynamic security sanitization and data releasability requirements.

This new dynamic climate raises fundamental questions about how the Department of Defense (DoD) goes about correctly integrating, certifying, and accrediting such configurations. This paper also identifies other hurdles in achieving successful MLS fieldings and suggests approaches for addressing them. The approaches are being investigated by the DoD MLS Program, and might lead to some significant changes in how DoD manages MLS capabilities.

Background

Lack of MLS within DoD computer systems has long been recognized as a significant shortcoming, because it limits interoperability and data fusion. To help address this problem, the DoD MLS Program was officially established in January 1990. The Defense Information Systems Agency (DISA) is the program manager, the Joint Staff is the central proponent for user

* This paper is primarily derived from work performed under Contract DAAB07-94-C-H601 for the Defense Information Systems Agency (DISA).

requirements, the National Security Agency (NSA) is the security coordinator, and the Defense Intelligence Agency (DIA) is the intelligence systems coordinator. The purpose of the DoD MLS Program is to expedite the fielding of MLS operational capabilities within DoD.

The DoD MLS Program was created because the marketplace for MLS products has been slow to materialize. In a sense, MLS technology insertion has stalled and needs a push to get it moving on its own. DoD has invested substantial funds to develop criteria and evaluate products; vendors also have invested substantial funds to create MLS products in response to DoD criteria, but the products still are incomplete or immature. Assistance from DoD is needed in marrying the products to DoD operational requirements and environments. The strategy thus has been to fund the DoD MLS Program to acquire and field near-term MLS capabilities. (The term *capability* means a collection of one or more products or components that can be integrated into a system to provide operational abilities.) The DoD MLS Program applies resources in focused attempts to integrate and field existing MLS capabilities, work out their shortcomings, and smooth the path for subsequent fieldings. By this injection, the objective is to both satisfy critical DoD MLS needs and stimulate the marketplace.

The DoD MLS Program strategy is to field the most promising products now as early operational capabilities in those applications and organizations where MLS needs are most acute, with resources focused on a small, common solution set. The strategy is to start fielding simple solutions; build incrementally to more complex capabilities; and expand from individual MLS components to common suites of components (e.g., incrementally add local area networks (LANs), database management systems (DBMSs), high assurance hosts, routers, and integrated communications security (COMSEC) products). Efforts are focused on the most mature products (e.g., those easiest to integrate, with effective vendor support). Much attention is placed on balancing security and useability. Much attention also is placed on seeing that the MLS capabilities fit easily into existing and near-term site architectures. For example, MLS workstations must be on a vendor platform already used by the site and must support common environments (e.g., Joint Deployable Intelligence Support System (JDISS)). New solutions are prototyped in laboratories. Then the concept of operation, security functionality, and the security design are worked out at a selected lead site. As capabilities are proven in the field, DoD MLS Program oversight can be phased out and widespread fielding of proven capabilities can proceed.

Near-Term MLS Capabilities

This strategy has been followed for several years and successful fieldings are ongoing. Currently the DoD MLS Program is focused primarily (but not exclusively) on four MLS solutions: the Command and Control guard, the Standard Mail Guard (SMG), two-level workstations, and the Operations (Ops)/Intelligence (Intel) interface.

- The Command and Control guard supports high-to-low and low-to-high flow between a command and control system and a lower-classified system. Essentially all transfers are fully automated, with guard review based on configurable context tables.
- The SMG supports two-way unclassified e-mail between classified and sensitive unclassified environments. The SMG is a joint effort of the DoD MLS Program and the NSA Multilevel Information Systems Security Initiative (MISSI).
- Two-level workstations are Compartmented Mode Workstations (CMWs) or other trusted workstations that are used to span two different environments (e.g., TOP

SECRET and SECRET). Normally the two environments spanned both operate in system high mode.

- An Ops/Intel interface is a CMW, placed in series with a filtering router, that can be used to span a security range greater than that which can be spanned by a CMW alone. The interface supports the key concept of an Ops/Intel user, who uses the workstation to operate in both environments. This solution is not limited to Ops and Intel environments. The only difference between a two-level workstation and an Ops/Intel interface is that the latter also includes a filtering router. From a user's view, both capabilities can be thought of as MLS workstations.

In general, these capabilities are either guards or workstations. Security guards are devices that mediate data flow between systems at different security levels. Both guards listed above provide fully-automated review of data flow and are built on high-assurance platforms.

Two-level workstations (and Ops/Intel interfaces) can provide richer functionality than guards. Because MLS workstations house the user interface, they can support an MLS operational view, even though the hosts and servers accessed operate system high. A user at an MLS workstation can be essentially a full-capability user of two systems at two different security levels and can transfer data between the systems. One MLS workstation thus replaces two system high workstations, while providing a look-and-feel similar to that of non-MLS workstations. Since a number of MLS workstations support open systems standards, this is a forward-looking architectural approach that provides MLS for standards-compliant applications and supports a migration strategy to increased use of commercial software.

Despite the promise of these solutions, there are significant hurdles in fielding such MLS capabilities. Some of the main risk areas are application integration, certification, and accreditation; all are discussed below.

Application Integration

Efforts to field MLS capabilities encounter the same integration problems as do other efforts (e.g., bugs in systems and applications, incompatibilities between applications, and lack of compliance with standards). Because MLS platforms enforce proper application behavior, they are not tolerant of applications that behave improperly. As a result, there are a variety of reasons why commercial applications might function incorrectly (or not at all) on an MLS workstation:

- Use of a license server, shared among multiple instantiations (at different security levels) of an application
- Use of temporary buffers shared among multiple instantiations of an application
- Use of disallowed system calls
- Attempts to seize the root window or the display color map

In many cases, applications can run on trusted platforms only if they are given privileges such that they are allowed to bypass the Trusted Computing Base (TCB). This is unacceptable from an accreditation standpoint. Some trusted workstation vendors maintain a list of applications purported to run on their trusted platforms, but do not point out that the applications must be given extensive privileges in order to run. In one case, a vendor claimed that an application was running with no privileges, whereas further questioning revealed that the application came with a license server that had to be given MLS privileges. Sometimes an application can be made to run

without privilege by reducing its capability. For example, in some cases a color application can be accommodated by forcing it to run in monochrome or with a reduced set of colors. Sometimes an application will run properly on an MLS workstation, but a new release of the application will not. Even minor application upgrades to seemingly harmless packages such as untrusted word processors can introduce subtle new vulnerabilities.

The best solution to these problems is to work with the vendors to have the commercial application software changed (for all users, not just for DoD). Vendors are sometimes reluctant to do this, but have proven willing when DoD can convince the vendor that the changes will increase the marketplace for the product. Sometimes it is not possible to change the commercial product and it becomes necessary for the integrator to write small trusted utilities (with limited system privileges) to perform simple MLS functions. An example of such a utility is one that mediates between different instantiations of an application that expects to communicate by shared software. In several cases, software developed by MITRE for such purposes has been incorporated by application vendors into subsequent commercial releases of the application.

There are a number of reasons to be optimistic that applications will become more compatible with MLS workstations. In some ways, MLS workstations function as though they were test machines that verify compliance with client-server environment (CSE) standards. Some of the technical characteristics that prevent applications from running on MLS workstations also would prevent them from operating in a CSE. As systems and applications evolve to CSEs, they will become more compatible with MLS workstations. In addition, now that MLS workstation sales are increasing, MLS workstation vendors, government laboratories, integration contractors, and others are establishing integration and porting laboratories to investigate which applications work on MLS workstations, which do not, and how the ones that do not work might be changed to work. This has the effect of both applying pressure on application vendors to upgrade their products and helping them to do so. Finally, the DoD MLS Program is planning to start a "registry" of applications that run acceptably on MLS workstations, with specific instructions on how to configure the applications properly.

Still, however, integration complexities will remain, due to applications that interpret standards differently, that use calls not allowable on secure systems, that exercise advanced operating system features not available on trusted workstations, or that simply do not comply with Application Program Interface standards or good programming practices. For the time being, substantial technical expertise is required to integrate applications onto trusted workstations.

In a broad sense, achieving a secure configuration initially remains difficult, but probably no more so than for past MLS fieldings. The significant new complication is the continuing need for MLS integration as software applications change. Where feasible, one way to address the problem is to limit fieldings to a common set of applications, to rely on central integration offices to perform secure integration, and to share information among integrators and certifiers. Fortunately, this strategy is consistent with strategies in several DoD communities (e.g., intelligence, command and control, counterdrug), which are selecting common sets of applications to create common operating environments. Work of the central integration offices can be overseen by central certification offices.

At a minimum, there must be central registration of fielded MLS capabilities, so that administrators can be alerted to problems and fixes. For some security guards, a program office can support central registration. CMWs might be registered to a central DoD office.

Certification

With both security technology and penetrator expertise advancing rapidly, much care is needed in certification. For MLS systems, effective certification is crucial. Even when the products in use have been rated by the National Computer Security Center (NCSC), much certification is needed to assess modifications to the rated product, to assess the integrated system as a whole, and to assess compliance with additional requirements such as integrity and denial of service requirements and requirements unique to the environment. Highly skilled specialized expertise is needed for the certification of MLS capabilities. These specialists must have the latest knowledge of attacks and vulnerabilities.

The challenge is to achieve effective certification with minimal resources. One way to do this is to define and manage a set of **common accreditation precedents**, with recertification required only if changes to the system, threat, or environment are sufficient to invalidate the original precedent. This not only reduces certification costs, but also permits thorough certification of the initial precedent by highly qualified people. This managed precedent approach best applies when there are common environments whose needs can be met by common security capabilities. Common accreditation precedents are discussed in the next section.

Initial certification must encompass the actual fielded configuration, to ensure that internal controls (e.g., UNIX parameter settings, security tables, application privileges) are properly configured. Proper configuration of the system can be as complex and important as designing an acceptable security architecture. A strong MLS design is of little use if the system is improperly configured. This is a crucial certification concern, because there will be strong incentives for integrators working under cost and time pressures to place more emphasis on getting applications running than on getting applications running securely, especially if there are not qualified certification personnel to review the configuration settings. Because of operational pressures to get new applications running and because even subtle configuration changes can undermine security, **there are real dangers that some MLS capabilities might give the illusion of MLS while in fact creating vulnerabilities that were not possible in system high systems.**

Certification has too often been narrowly viewed as something that happens *before* a system is placed into operation. But certification does not end when the system becomes operational. Changes demand *continuing* certification as part of configuration management. Unfortunately, field security officers typically are not technically qualified and do not have the time to certify fielded (and changing) MLS capabilities. Automated security assessment tools are critically necessary to ensure that system security is maintained after the initial certification. Fundamental changes to the certification process are needed so that certifications require an in-place tool infrastructure that can accommodate continuous system changes.

Commercial and freeware tools are available that assess generic systems (e.g., UNIX). These tools, such as Security Analysis Tool for Auditing (SATAN), also enable inexperienced people to launch automated attacks on systems. The existence of such tools to aid attackers makes it even more critical that the tools first be used by defenders to test defensive fortifications. MLS systems require tools that assess for proper MLS configurations (e.g., they should check program privileges, user authorizations, network configuration). Such tools will require careful design and control, to protect against disclosure of the tools and to minimize the impact should disclosure

occur. The tools need to be run on a regular basis to test system defenses from the inside and outside and should check for known vulnerabilities.

Internal tools (i.e., run from the systems they are checking) can be built to expand or supplement tools that already come with MLS workstations or third-party tools such as Security Profile Inspector (SPI). *External* tools (i.e., can be built to expand or supplement commercial and freeware sources. An infrastructure must also be in place to update the tools frequently, as new problems and fixes arise. External tools might be viewed as network test tools and internal tools as configuration verification tools. Much work is needed to expand and improve current tools and facilitate their use by people who are not expert on system internals.

Different tools and procedures are needed to ensure proper certification of *guards* that are changed in the field (e.g., to ensure that a new type of data is not released or to begin allowing release of certain data to a participant). Such changes are necessary to satisfy dynamically changing mission needs, especially in missions involving multiple security environments (e.g., different nations). Since modern military missions evolve quickly, changes must be made, tested, and implemented in the field. Guard needs are different from MLS workstation needs, in that guard changes are more likely to be limited in scope (e.g., to a few sites or for a short time interval). Tools to implement and certify these changes must be fielded before guards can be modified in the field. The tools should be approved in advance by the accreditor; clear rules are needed on what procedures must be followed in making changes and what can and cannot be changed.

Accreditation

Currently, DoD Directive 5200.28 provides general guidance on how to apply MLS technology to different environments [1]. However, expert interpretation is needed in applying this guidance where it has not been applied before. Furthermore, the guidance is general and leaves many technical questions unanswered. It has often been stated that there is a need for improved guidance that is more clear and less subject to misinterpretation. In fact, a good case can be made that DoD already has such guidance in the form of accreditation precedents, which represent a sort of *Case Law* for DoD MLS. Serious consideration should be given to using precedents (rather than high-level policy) as the primary bases for accreditations. This requires further experience with approaches for establishing and managing common accreditation precedents. The DoD MLS Program is beginning an effort to establish and manage several common accreditation precedents, through efforts to field common MLS capabilities. These fielding efforts are providing lessons learned that might change how security is managed within DoD.

Some of the lessons involve *establishing* the common accreditation precedents. Central to this process is the pairing of specific MLS capabilities to common environments. This pairing is necessary to ensure adequate analysis (via a real, detailed, worked example) of whether a capability provides sufficient protection for a particular environment. Examples of common capabilities established by the DoD MLS Program are the near-term MLS capabilities summarized above under that heading. A primary lesson learned from establishing common precedents is that environmental differences complicate definition of precedents that truly are common. Typically, two or (preferably) three different fieldings are needed before a common precedent can be fully established. These fieldings gradually add greater functionality and capability and address the operational needs of different environments. They also clarify which

site differences can be expected (e.g., different common operating environments) and which truly are site unique (e.g., locally developed software).

Another interesting lesson learned regarding establishment of precedents is that the initial precedent is not based on specific requirements. Rather, the precedent is constructed by a highly subjective risk management process, involving much interaction with accreditor(s). Once the precedent is established, it can be used as a requirements baseline for subsequent fieldings. This is an interesting finding, in that certification does not apply in its traditional sense. That is, certification is usually defined as determining compliance with requirements. Without a requirements baseline against which to measure, certification is more subjective. The complexity and subjectivity involved in establishing and certifying common accreditation precedents leads to the conclusion that the people involved must be the top security engineers and accreditors available to DoD. While objective guidance surely can be used to assist the creation of new precedents, much of the process is inherently subjective.

The second area of lessons learned involves *managing* the common accreditation precedents. Once a precedent has been established, subsequent fieldings are assessed against the precedent, with a determination made of whether the precedent still applies. The accreditor must decide whether a follow-on fielding is a variation or a new precedent. Sufficient changes in the environment, system, or threat might call for a new precedent. Even when follow-on fieldings of a common capability are performed, they still will bring incremental changes to the initial accreditation precedent. This increases the complexity of managing the precedents.

New Precedents for Operating Modes and Assurance

Two important aspects of the common MLS capabilities currently being fielded are that (1) they do not operate in what has traditionally been thought of as the MLS mode of operation, and (2) they take non-traditional approaches towards achieving overall system assurance. In other words, the precedents establish new ways of thinking about modes of operation and assurance for MLS capabilities. The new ways of thinking are summarized below. These precedents illustrate that guidance based upon fielded common capabilities is a necessary supplement to guidance based on generic, high-level policy.

Sample Precedent: MLS Workstation

Most vendor MLS workstations are targeting class B1 (or CMWs). Initially, a concern was raised that most MLS environments need class B2 or higher protection, in accordance with DoD Directive 5200.28, enclosure 4 [1]. It is important to stress, however, that B1 MLS workstations as applied by the DoD MLS Program do not violate the DoD Directive. The Directive applies to many situations in which, for example, SECRET and TOP SECRET cleared users directly access the same system and might even have direct access to the operating system. In contrast, the MLS workstations discussed above do not support direct or indirect access by users cleared at different levels. Instead, all users with access to MLS workstations are cleared to the highest level of data processed on either system.

The point here is that the DoD policy does not explicitly apply to the environments in which many MLS workstations are being used (i.e., where workstations support simultaneous access to existing legacy systems at different system high security levels, but do not permit MLS use of the workstation itself). In the case of the Ops/Intel interface, this mode of operation is referred to as

system high (special case). MLS capabilities are the goal, not the MLS mode of operation. This is not a trivial semantic distinction. The MLS mode of operation still is needed for some MLS DBMSs, servers, and other capabilities, but it clearly is not the only way in which MLS capabilities can be used.

Sample Precedent: Guards

Guards offer a way of achieving assurance that is different from that described in the Trusted Computer System Evaluation Criteria (TCSEC) [2]. That is, guards represent the use of multiple, independent checks as an alternative to relying upon one check. Philosophically, this is similar to the separation of duties used in banking applications and the two-man control used in military applications. So, though different from the TCSEC approach for assurance, it is not a new approach. Underlying the use of guards is the presumption that a physically separate, independent check, taken together with checks provided within the connected systems, provide a degree of assurance beyond that of either component by itself.

Sample Precedent: Ops/Intel Interface

This is an important new accreditation precedent; it represents a new way of interpreting security policy that broadens the scope of CMW applicability. It combines a DoD-originated solution (i.e., a CMW) and a commercial solution (i.e., a filtering router or commercial firewall) in series to achieve greater security than either solution could provide by itself.

Conclusion

Current MLS fieldings are different in kind from early fieldings and are establishing new ways of thinking about MLS. MLS capabilities still require careful security engineering and integration and are not turn-key capabilities. Adequate resources must be budgeted for application integration, certification, and accreditation. Central registration, automated security test and configuration verification tools, and common accreditation precedents are needed to oversee MLS capabilities that change dynamically in the field.

Recent technological advances have made some MLS technology ready for application in the field. A significant number (i.e., 10-20) of improved or new MLS products are becoming available that appear to overcome many of the shortcomings of past products. A window of opportunity is opening. Government and industry must work together to take advantage of this opportunity, and must refine and apply MLS capabilities in order to establish a broader marketplace for MLS. The promise of MLS technology has never been higher.

Acknowledgments

This paper benefits from ideas and experiences contributed by COL Joe Sheldon, LTC Joe Alexander, Pete Kurzenhauser, and Charley West of DISA; by John Seymour of the Joint Staff; by conference reviewers; and by Kate Arndt, Jeff Berger, and Gary Huber of The MITRE Corporation.

References

[1] DoD, 21 March 1988, *Security Requirements for Automated Information Systems*, DoD Directive 5200.28, Washington, DC.

[2] DoD, December 1985, *Department of Defense Trusted Computer System Evaluation Criteria*, DoD Directive 5200.28-STD, Washington, DC.

APPLYING COMPUSEC TO THE BATTLEFIELD

Diane M. Bishop and Stephen R. Arkley

Computer Sciences Corporation
1301 Virginia Drive, Fourth Floor
Fort Washington, PA 19034

Abstract

In the past, Army tactical automated information systems (AIS) were exempt from the stringent security requirements imposed by regulations. As tactical systems evolved from single purpose stovepipe systems to networks of systems, additional security is required. For many tactical systems, the application of strict and inflexible security requirements may be excessive and restrictive. Security requirements for tactical AIS often inhibit the ability to complete the assigned mission in a timely and user friendly manner. Systems are currently being developed and certified using the DoD 5200.28-STD [2], Trusted Computer Security Evaluation Criteria (TCSEC) and NCSC-TG-005 [5], Trusted Network Interpretation (TNI) standards. The application of these standards to tactical systems should be more flexible to allow the Designated Accreditation Authority (DAA) the ability to implement the security measures which allow the system to fulfill mission requirements while providing an adequate level of security. The security requirements should be developed using the TCSEC and TNI criteria as the initial input to the security engineering process. Refinement of requirements to fit the tactical mission is then done based on a risk assessment. Certification should be an evaluation of the system's performance against these developed requirements, rather than an evaluation against a TCSEC rating.

Keywords: Computer security, tactical, access control, battlefield, AIS, requirements, TCSEC.

Introduction

AIS and communications systems with AIS features are proliferating the tactical battlefield. There is a bonafide need for most of these systems to interface, either full time or part time, with other systems. Current security doctrine dictates the security requirements and procedures based solely on user clearances and the type of information being processed. In many cases, the TCSEC class resulting from applying the TNI and CSC-STD-004-85 [6], Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments (Yellow Book), are based on a network containing a broad base of users with different needs-to-know, clearances and access rights. This current methodology does not provide any flexibility to the DAA to use the TCSEC requirements as a starting point and, through a pragmatic approach, adjust the specified security requirements to counter the specific risks associated with their system. This type of common sense approach would save scarce budget dollars by validating the need for each of the security requirements as well as the identifying requirements which are counterproductive to the mission.

The application of security requirements dictated by the TCSEC, regardless of actual need, may jeopardize the timely accomplishment of the tactical mission. As the multitude of worldwide and widely diversified networks are connected, it is easy to rationalize the need to apply the requirements of the A1 security evaluation class to everything. The blind application of security requirements without considering the mission and risks during the engineering process could result in excessive cost, wasted time and adverse mission impact. This paper examines some of the key considerations when conducting a pragmatic assessment of the level of security needed in a tactical system, and discusses an alternative to the current method of identifying requirements and certifying AIS.

Background

Much of the current security guidance was written in the 1980s (e.g., TCSEC, 1985; MIL-STD-2167A, Defense System Software Development, 1985; NSDD 145, National Policy on Telecommunication and AIS Security, 1984). When this documentation was written, many tactical systems were exempt from the security requirements imposed on non-tactical systems (e.g., Army Regulation (AR) 380-380 exempted Army Battlefield Automated Systems (BAS) from requirements contained in that regulation). However, in August 1990, AR 380-380 was superseded by AR 380-19 [3], which no longer exempted BAS from these requirements.

This was a major change to the manner in which security for tactical systems was applied. AR 380-19 requires all Army systems, with the exception of dedicated systems, to at least meet the requirements of the C2 security evaluation class contained in the TCSEC. In accordance with AR 380-19, only Major Command (MACOM) commanders or the Administrative Assistant to the Secretary of the Army are authorized to approve exceptions to the TCSEC requirements.

COMPUSEC Considerations in Requirements Definition

Tactical systems are unique in that they normally serve a specific, limited range of functions and are used under conditions which demand the user be able to quickly, accurately and easily use the system. Conditions such as darkness, extreme cold, and high stress make these demands difficult to meet. For example, if it is necessary for the user to enter a user ID and a password in a time-critical situation, the system is not supporting the mission.

The current method of determining security requirements does not take any mission requirements into consideration until well into the accreditation process. Requirements are currently determined by using the Yellow Book and TNI to calculate a security evaluation class which consists of a predetermined set of requirements listed in the TCSEC. This set of requirements is applied against all systems, regardless of size, equipment type, or mission. The certification process is used to evaluate the system against this predetermined set of requirements. If a requirement is not adequately met, because it adversely impacts the mission and therefore is not fully implemented (if at all), it is identified during the certification testing. A residual risk analysis, which considers the system threats, vulnerabilities and the mission, is then performed to quantify the level of risk associated with the requirement not being fully met. If the risk is determined to be too high, countermeasures must be applied to mitigate risk. Because the requirement was not fully met, the system cannot be certified to meet the TCSEC

security evaluation class. Although the selected countermeasure may mitigate the risk, it does not satisfy the requirement (e.g., the use of physical security to mitigate the risk of not having password protection). Therefore, in deciding to accredit the system the DAA is obligated to accept any residual risk.

If mission requirements are considered during the engineering assessment, rather than during the accreditation process, any TCSEC requirements which have an unacceptable impact on the tactical mission would be identified early in the process and excluded. The risk associated with not implementing that requirement would be determined as part of the threat/vulnerability analysis which is performed during the engineering assessment. If the risk appears to be too high, other more acceptable security measures (either manual or automated) would be applied. During certification, that requirement would not be evaluated since it had been excluded. This would allow the DAA to tailor the security requirements to support the tactical mission and certify the system against the developed requirements rather than against the predetermined requirements mandated by the TCSEC security evaluation class.

The following paragraphs discuss the major considerations in applying this practical approach.

Purpose and Use

The intended purpose and use of the system must be identified in the greatest detail possible in order to provide accurate information for use in the rest of the requirements definition process. In an effort to minimize unnecessary expenses, many Services are calling for systems which can be used both in a tactical environment and in garrison. Many of the high-tech systems require constant hands-on training in order for the user to stay proficient. This means that the system is exposed in multiple environments to many types of users. Knowing this during the design phase is important so the appropriate security can be incorporated into the design rather than added on as an afterthought.

Details pertaining to the system's purpose and use will also provide the type of information in the system (classification, caveats and handling restrictions) and the users' status (e.g., clearance, need-to-know). This information can be used to perform the initial requirements definition.

Initial Requirements Definition (Red/Yellow Book)

This initial requirements definition, using the TNI and Yellow Book, is based on the maximum classification and number of categories of the information in the system, and the minimum security clearances of the personnel using the system. The initial requirements definition results in the determination of a mode of operation and calculation of the TCSEC security evaluation class and associated security requirements.

For example, a portable electronic key management device is being developed in an open environment which will be used by personnel who are cleared to the SECRET level ($R_{min}=3$) and which will process SECRET information with two categories, with one containing SECRET data and the other containing unclassified sensitive information ($R_{max}=4$). Using the Yellow Book, we determine the requirement is for multilevel security mode of operation and a B1 class of security. The security evaluation class and associated security requirements will be used as the initial input for further developing the security requirements.

Threat Environment

The threat environment is one key consideration when developing the security requirements for a system. The threats a tactical system will be exposed to in a garrison environment will differ from those encountered when it is deployed in a tactical environment. For systems which will be used in both environments, the greater threat environment should be used to develop security requirements. The threat to a tactical system in a deployed environment is very difficult to determine. The location and method of employment may be different each time the system is deployed. The known and postulated best-guess threat must be determined by the intelligence community based on experience. The threat in a garrison environment is more easily determined, since the location and method of employment are known. For example, a computer used to maintain unit logistics information may be used at Fort Campbell, Kentucky in a Supply Officer's office; however, when deployed it may be mounted in a vehicle and airlifted to an unknown area of operation.

The threat information must be as detailed as possible based on the intelligence communities' broad base of information. Often this information is highly classified; however, it is critical that accurate, detailed threat information be used in conjunction with equally detailed system vulnerability information. The threat information should be validated by the appropriate agency (e.g., Intelligence and Threat Assessment Center (ITAC) for the Army) prior to use.

System Vulnerabilities

System vulnerabilities must be analyzed from all aspects of the system's employment. The vulnerabilities in garrison as well as a tactical environment must be addressed. With the movement toward seamless networks, one of the key areas to analyze is the network and its composition. Situations such as the possibility of covert channels, connections which are in violation of an established treaty, or connections to another U.S. system which has unacceptable risk, must be prevented. It is conceivable that an adversary could connect to and exploit a system by working their way through several other systems. An adversary will follow the path of least (security) resistance to gain access to what they are targeting. Another potentially devastating vulnerability may exist in the system's susceptibility to the introduction of a virus, Trojan horse or logic bomb.

Some vulnerabilities may be very obvious, while others may be more difficult to determine and properly document. A system's vulnerability to Electromagnetic Pulse (EMP), which could result in denial of service, would need to be identified during the engineering process. On the other hand, the vulnerability to a loss or compromise associated with the use of a removable hard disk drive for the storage of classified information is better known and understood.

Risk Assessment

Using the information resulting from the threat and vulnerability analysis, an accurate assessment of system risks can be made. This assessment explores the probability of an identified threat attempting to exploit a system's vulnerability, the likelihood of success, and the postulated ramifications if the vulnerability is exploited. The risk assessment is most accurate if a specific threat is associated with a specific vulnerability. Even if there is no known threat, a significant vulnerability should be treated as though an adversary will exploit it. This step in the

engineering process may also identify/justify areas where security requirements may be reduced.

For example, suppose an adversary has the capability (high probability of success) and desire (high probability of an attempt) to intercept data by tapping into fiber-optic lines (threat). The system design does not prevent the tap or notify the user that a tap has been installed (vulnerability). Because of the use of Communications Security (COMSEC) devices by the system, the data will be encrypted and therefore unreadable by the adversary (minimal ramifications of success). Based on this analysis, it may be determined that the risk to the system is low in this particular case.

The risk assessment results are part of the basis for justifying the requirements.

Requirements Definition

A system, regardless of type, will only be used if it is reliable in terms of timeliness of information delivery and ease and speed of use. The implementation of security requirements should be as transparent as possible to the user, while fully supporting the mission and providing adequate security.

Developing the system requirements is a critical step in the engineering assessment process. This engineering process correlates system risks, mission needs, TCSEC requirements and technical considerations, and will be used to develop and justify the requirements needed. In a case where high security is needed in a garrison environment, but those requirements adversely impact the mission when deployed, the capability for authorized personnel to turn those features on and off may satisfy the needs for both operating environments. The requirements developed in this step of the engineering assessment, rather than the TCSEC requirements, will be evaluated during the certification testing. This gives the DAA the flexibility to develop requirements which best support security and the mission, and gives the engineers the ability to build a system which is user friendly and less costly.

The engineering assessment process feeds into the certification and accreditation process. Figure 1 shows the relationship between the engineering assessment process and the accreditation process.

Certification Testing

Certification testing is part of the accreditation process and comprises the formal evaluation of the requirements. In this approach the certification testing would evaluate the *developed requirements*, which consider the mission and system risks, rather than the list of requirements in the TCSEC. During this comprehensive evaluation, the technical and nontechnical security requirements of the system are thoroughly tested and evaluated. This establishes the extent to which the system design and implementation meet the specified set of security requirements. After completion of the certification testing, a certification report is generated and included as part of the accreditation documentation.

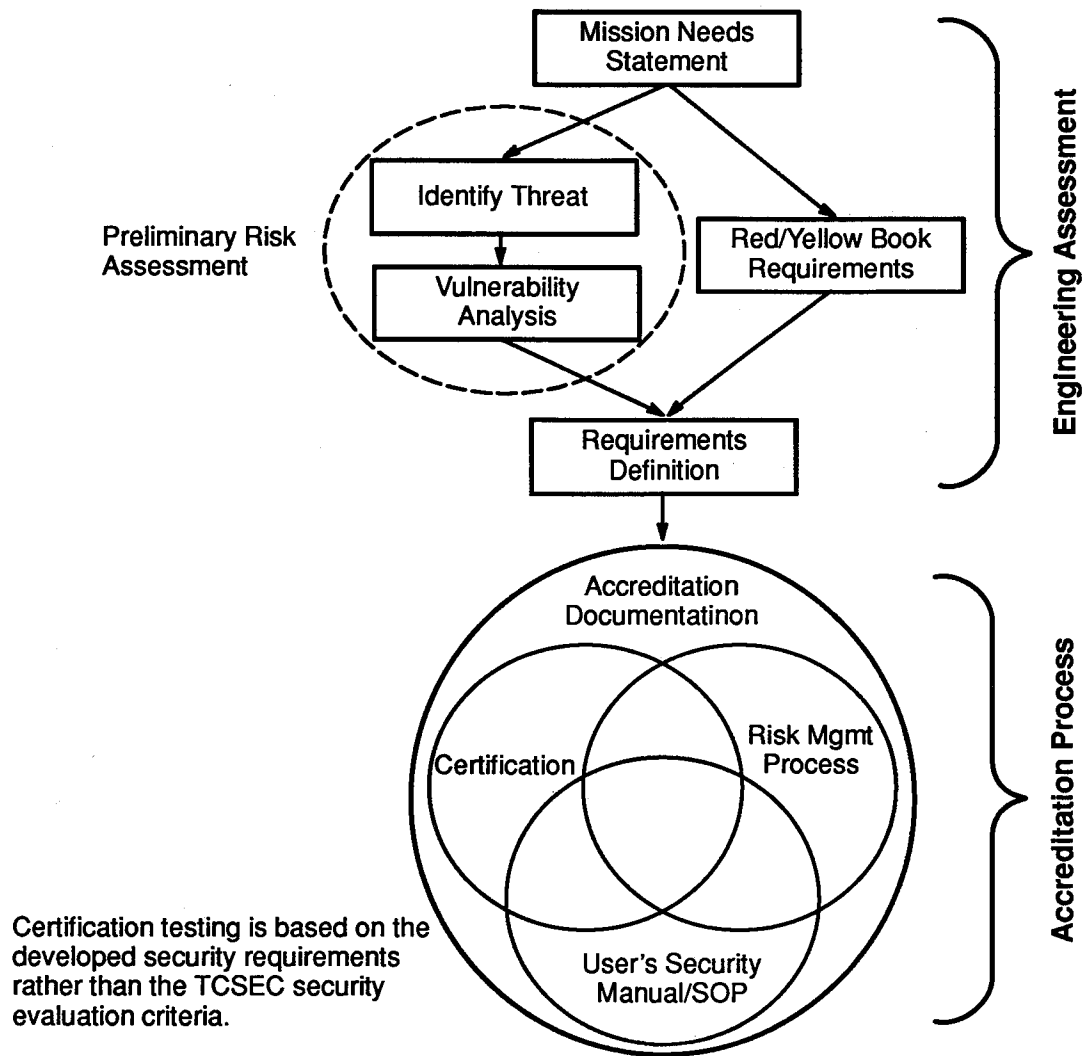


Figure 1: Requirements Definition Process

Residual Risk Analysis

The residual risk analysis, which identifies the risk that remains after the implementation of the requirements, is an important part of the risk management process. These results include the risks associated with system requirements implementation, and takes into account threats and vulnerabilities outside of the system requirements. For example, there may be risks based on how well the procedural security responsibilities of the Terminal Area Security Officer (TASO) and the users are executed. A detailed assessment of all risk must be made during the risk management process. As shown in Figure 1, the certification testing, User's Security Manual and the risk management process support each other in an iterative process designed to identify and reduce risk. Using the results of the certification testing as part of the input to the risk management process, a residual risk analysis is conducted to determine what risks remain and at what level. The results of this analysis are formally documented and included in the

Security Plan/Accreditation Documentation for the DAA's review. If the residual risk is determined to be acceptable, the accreditation is approved. If the residual risk is too high in the DAA's opinion, additional security measures are selected to mitigate those risks to an acceptable level.

The security requirements should be balanced between acceptable risk (DAA's opinion) and mission demands. Generally, the mission is not fully supported if the level of computer security is inadequate to protect sensitive or classified data. Conversely, if there is an excessive level of security, the mission support level could be low since the security features may hinder mission progress. When considering risk, if there is very little computer security, there normally is high risk. If there is excessive security, the interrelationship of the security features may introduce additional risk. The optimal level of security provides the most mission support while reducing the risk to the lowest reasonable level. Figure 2 shows the interrelated impacts of risk and mission support on the optimal level of security.

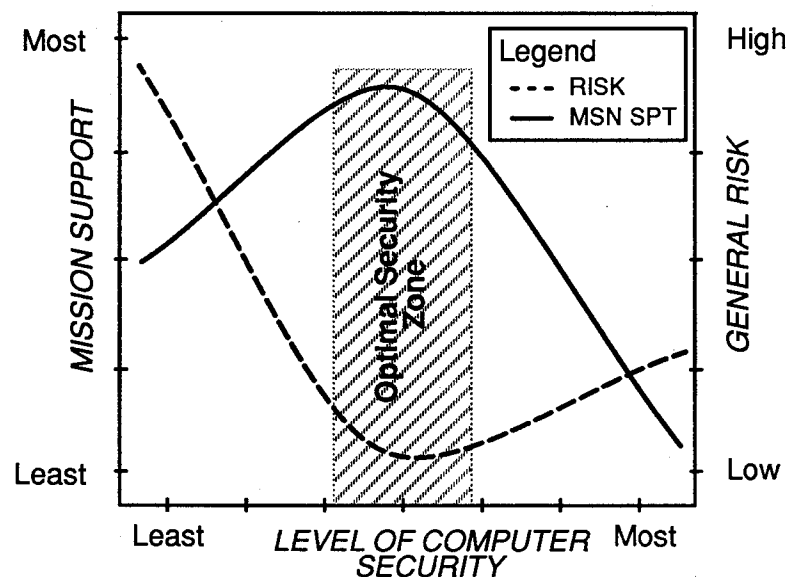


Figure 2: Security—Balance between Risk and Mission

The decision to have more or less security, based on the demands of the mission, ultimately rests with the DAA. The method each DAA uses to determine risk, and what is to be considered an acceptable level of risk, will vary between DAAs. This is a result of different levels of experience and training among DAAs, an inconsistent quality of support/advice received, and the use of numerous risk assessment methodologies. The resulting wide range of risk analysis results is a known problem; however, with limited experience in global networking, specifically the interconnection of vastly different networks of networks, the current way of performing risk analyses is the best available. The solution to bringing the results of the DAAs' risk decisions closer into line with each other rests in the continued improvement and consistent application of advanced security technology as well as the development of a standard set of security methodologies. These methodologies should be included in a single standard which serves as the security methodology reference. This reference should include a detailed description of the

approved methodologies for use in the engineering process and the accreditation process (and their subprocesses).

COMPUSEC on the Tactical Battlefield

Considering the mission needs and developing system requirements before beginning the accreditation process has several key advantages. These advantages include:

- DAA flexibility to tailor the system to best support the mission.
- Justified set of security measures.
- Cost and time savings.
- Less chance of incurring new risks associated with implementing a requirement which adversely impacts the mission or is unnecessary.

Examples of requirements which may not be justified are the need for a trusted path or a login sequence for a device which is used in an environment where physical protection is provided by the user (e.g., digital photographic camera system); the need for an audit trail where the user is known and is authorized to execute all functions the device is capable of performing (e.g., Forward Entry Device); or the need to have the human-readable output on a Liquid Crystal Display (LCD) labeled SECRET when access to the device is limited to personnel with a SECRET clearance and a need-to-know. In extreme cases, members of the Armed Forces would be required to log on and enter a password despite operating under conditions which may make these tasks next to impossible. Audit requirements may slow the system's speed and response time to an unacceptable level.

The need for and level of each of the security requirements should initially be justified based on the TCSEC security class, and then further developed using the results of the risk analysis and mission requirements before beginning the accreditation process.

Cost savings could be achieved by not wasting time or resources implementing unnecessary requirements. This reduces the time and expense of producing programming code, designing hardware, writing associated documentation, and testing features which are not needed.

For each requirement that is implemented, there is the chance that additional risk may be incurred. For example, if a network has the (unjustified) requirement to collect audit data, there is a requirement for audit management. This could entail the transmission of audit data to primary and alternate collection points on a scheduled basis. This exposes the information to possible compromise and uses part of the communication media's bandwidth which may have been better used to transmit mission data. It also means that a system administrator must manage additional files, take time to review the audit logs, and take appropriate action if a discrepancy is discovered. If the requirement for audit was identified as counterproductive to the mission during the engineering assessment, and therefore not specified as a requirement, none of the additional risk associated with maintaining audit information would need to be assumed.

Designing tactical systems for a specific purpose helps reduce the requirement for extensive security. If a system is designed to send field artillery target data, and the information is

preformatted by the system, the risk of an adversary exploiting the network is remote. Using communications security techniques, the data can be encrypted (data confidentiality), authenticated (data integrity) and the network can be designed to remotely disable (disconnect) the system in the event of capture or malfunction. These protective measures provide a great degree of protection for tactical systems.

Given a tactical system which has been designed for a specific purpose, the two major security requirements should be the ability for the system/user to authenticate and the ability to isolate the system from the rest of the network. The purpose of the authentication is to ensure the data being received is legitimate. If a device begins sending erroneous data, the network should immediately know. The ability to isolate the device from the network prevents an adversary from overloading the system with erroneous data or exploiting the system. The need for data confidentiality may or may not be a major consideration. The ability of state-of-the-art COMSEC devices make this an easier requirement to fulfill.

Conclusion

The approach discussed in this paper defines security requirements using the TCSEC as a starting point. The TCSEC requirements are then adjusted based on the mission, threats and vulnerabilities. The resulting set of developed requirements are then used in the accreditation process. Certification testing evaluates how well the system meets the developed requirements rather than the TCSEC list of requirements. This approach gives the DAA the flexibility to fully meet the mission requirement; while providing adequate security, cutting costs and reducing the potential for creating additional risk. This approach is based on the application of interpretations of AR 380-19 and the experiences of various contractors and government agencies gained during the development of tactical automated information systems and communications systems with AIS features—in particular, the systems used in the Army Tactical Command and Control System (ATCCS).

References

- [1] DoD Directive 5200.28, Security Requirements for Automated Information Systems (AISs), Department of Defense, 21 March 1988.
- [2] DoD Standard 5200.28-STD, Department of Defense Trusted Computer System Evaluation Criteria, National Computer Security Center, December 1985.
- [3] AR 380-19, Information Systems Security, Headquarters, Department of the Army, Washington, DC, 1 August 1990.
- [4] AFR 205-16, Computer Security Policy, Department of the Air Force, 28 April 1989.
- [5] NCSC-TG-005 Version-1, Trusted Network Interpretation, "Red Book", National Computer Security Center, 31 July 1987.
- [6] CSC-STD-004-85, Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments, "Yellow Book", Computer Security Center, 25 June 1985.

The contents of this paper are not necessarily the official views of, or endorsed by, the U.S. Government, Department of Defense, Department of the Army, Program Executive Officer Command and Control Systems, or the Army Communications-Electronics Command (CECOM) and Fort Monmouth.

SECURITY REQUIREMENTS FOR CUSTOMER NETWORK MANAGEMENT IN TELECOMMUNICATIONS

Vijay Varadharajan
Hewlett-Packard Labs.
Filton Road, Stoke Gifford, Bristol
U.K.

1 Introduction

The Customer Network Management (CNM) is an emerging standard defined by the CCITT to allow telecommunications service subscribers, i.e. the customers of telecom organization (TOs), on-line access to the network for the purpose of management. It is different from the conventional method used by the telecom operators where complaints are made via telephone, and bills are sent via mail. CNM allows telecom organizations to extend the subscribed telecommunication services into the customer premises. This will enable the customers to link this information to their private network management system. CNM allows customers to perform a wide range of activity such as inventory management, service ordering, accounting management, amongst others. The future trend is that CNM is going to become increasingly important particularly in the telecommunications market, where there is a greater need for giving customers access to information such as how much they are paying for and for what services.

This paper considers the CNM application and outlines possible security requirements that arise in this environment. The aim of this paper is to clarify some of the assumptions and point out some of the security threats that can arise, and propose the type of security services and mechanisms that are necessary. We look at some of the design choices. We do not describe protocol details in this paper. They form the subject of another paper in preparation.

2 CNM Overview

The main aim of CNM is to support integrated management of the entire portfolio of Telecom supplied services, within the customer network. In addition, the customer's network also consists of a complex range of equipment, which are components of the end-to-end network service required by the customer to achieve some defined business objective [1].

The customer network can be considered to consist of a connectivity communication infrastructure services, an infrastructure of equipment - voice and data equipment as well as desktop and host computer systems -, and supporting system software and end-user applications.

CNM capabilities are intended primarily to enable customers to manage their portfolio of Telecom services. To get a better idea on CNM capability, let us briefly consider inventory, service ordering, and accounting management in this context.

CNM Inventory Management allows the customers to maintain information about the Telecom supplied items that the customer subscribes to. These items include equipments such multiplexers and PBXs, communication circuits such as voice tie-line between 2 PBXs, and other miscellaneous items such as modem cabinet. The customer is able to view and change some attributes and relationships related to the inventory items that he uses. Furthermore, information on these supplied items can be merged with information on the customer's private inventory items (e.g. LAN, Host) to provide a complete inventory picture related to the customer's business.

CNM Service Order Management allows the customer to perform on-line ordering of new services, modification of existing services on a "need to" basis. Status of a service order request can be tracked automatically by this CNM function.

CNM Accounting Management provides a customer with on-line and timely delivery of invoicing and telecom usage information on the services he subscribes to. This function can also provide analysis, processing and reporting capabilities on a range of invoicing, telecommunications and financial usage topics. The customer is able to get a better understanding on the amount he is paying for the services, whether he is charged for services that belong to other customers, and whether he has over-subscribed a certain service.

2.1 System Diagram

The platform upon which the management solutions can be built in the area of CNM includes the following :

- Communication Infrastructure.

This provides distribution-transparent access to objects.

- Presentation infrastructure

A user selects the managed object on the graphical map and executes commands through pull-down menus that are controlled by windows.

- Data Relationship Store

This is a repository of information which comprises managed object instances as well as relationships (linkages) between these instances. The information can be manipulated via management applications.

- CNM applications

In general CNM applications will also need to access other data NOT managed by the platform and as such accesses to such data may or may not be controlled by access control mechanisms outside that of CNM.

- **Development Tools**

Assist developers to deliver solutions with a minimum of effort.

2.2 Usage Scenarios

There are two types of CNM systems : (i) Customer Premise Station (CPS) and (ii) Value Added Integrator (VAI). This is illustrated in Figure 1.

The CPS is the CNM system installed at the customer's location. A user of CNM may be an employee of the TO's customer or may be an employee of the TO itself. When using CNM applications, a user would probably access both organization (whether customer or TO) private data and TO provided data.

Typically TO provided data resides on the Value Added Information (VAI) whilst organisation private data resides on the Customer Premise Station (CPS). TO provided data MAY reside on the CPS because of performance or cost considerations.

Typical usages include :

1. A user at the CPS accesses information in the Customer Private database using a CNM application.
2. A user at the customer site accesses information in the Customer Private database via "local" customer network.
3. A user in (1) or (2) above accesses information from the the VAI owned by the telecom service provider - TO operator.

3 Analysis

3.1 Security Threats

Consider the situation where a user U through CNM application A in machine X requests for information stored in a database through application B in machine Y. In our discussion, the application B in Y is in fact the remote DRS. DRS is a trusted component; in particular, we trust the DRS for access mediation to the VAI.

Let us now consider the important threats in this environment. These are a subset of possible threats given in[2].

- *Masquerading* : A user U' via some application A pretends to be user U to the DRS (local or remote).
- *Unauthorized Access* : Unauthorized access to information in remote DRS.

A user should only be able to access the information he has permissions for from allowed systems.

Information in the remote DRS (in Y) is accessible to anyone who has root permissions or the database administrator capability. Furthermore, as third party products other than DRS may be involved, secure access to data cannot be guaranteed if proper access controls are not enforced. Thus there is a responsibility on local management as well as in the CNM platform.

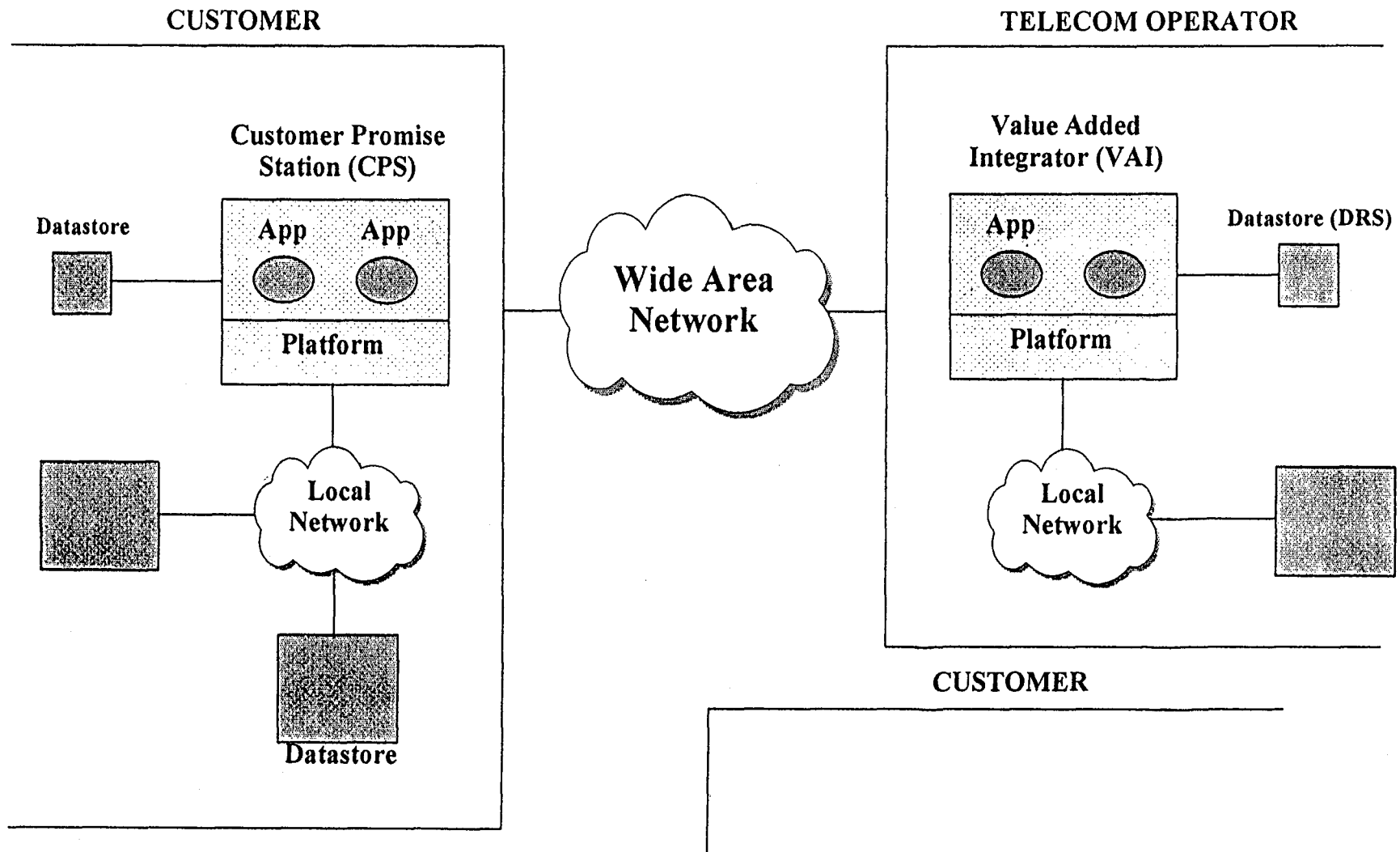


Fig 1: Customer Network Management Environment

- *Communication Security Threats*

1. Modification of messages between machines X and Y thereby affecting the contents of the request from A to B, or the results from B to A.
2. Eavesdropping on the messages from machines X to Y.

3.2 An Example

These security threats were then explored in a practical situation, in the context of a Telecom provider requirements. This led to an informal classification of the perceived threat as HIGH or LOW given below. (*Note that this is just one example and is not intended to be representative of a wider set*).

- Masquerading : HIGH
- Unauthorized Access to DRS : HIGH
- Modification of Messages : HIGH/LOW
- Eavesdropping on Messages : LOW

3.3 Architectural Considerations

Before we discuss the required security services, let us set out some assumptions about the platform and its consequences for the appropriate architecture of security services.

We assume that security is to be enforced as far as possible through the platform itself and not through the management applications that use it. This implies that the target DRS should be responsible for guarding against masquerading and unauthorized access, and it should be the responsibility of the user/calling application to present proof of identity to the DRS. However there is no need for the user/calling application to interact directly with the target DRS. The task of getting proof understandable to the DRS can be delegated to the authentication service, which is distributed across all the mutually aware CNM hosts.

If the management application is not to be trusted to carry out authentication on behalf of the target DRS, it should neither be trusted to make access decisions. Hence an application may make use of function masks for local restriction of the user, but this is not part of the security solution of the platform that we envisage.

We will also assume that the access mediation is carried out by the DRS. That is, the access policy resides within the DRS and that both access decision and enforcement are carried out by the DRS. Other alternatives are possible. For instance, we could envisage a separate authorization server component that acts as a repository for access policies. The user/calling application can obtain from this authorization component the relevant privileges, and present them along with the proof of identity to the DRS. Using these two pieces of information, the DRS can grant or deny access to its information. We will not be discussing such an approach in this paper.

4 Security Services

This section addresses the security services which are required to counteract the security threats mentioned above.

4.1 Against Masquerading

To counteract the threat of masquerading, we need to ensure that the user claiming to be U is actually U, and that the request comes from the claimed machine X.

There are essentially two approaches to achieving this:

1. Target DRS trusts the local authentication service at the initiating end to have authenticated the user U.
2. Direct authentication of user U by the target DRS in machine Y.

As mentioned above, the first approach is achieved using a component of the authentication service in each CNM machine. The initiating application uses the authentication service to establish the user's identity. This authenticated information is then presented to the DRS which can then make its access control decisions based on this presented information.

In the second approach, the target DRS does not believe the authentication process carried out at the initiating end.

In the CNM environment, in any case we have to assume that the CPS administration takes proper care of its secrets. So long as this is true, the PTT CNM station can guarantee protection for data relating to a customer, and to protect against masquerading amongst CPS users. Hence we believe that the first approach is more appropriate.

In fact the guarantees are "contractual" in nature. In this respect, the PTT does not trust the customer in order to protect its own interests, but has to make assumptions about customer behaviour - which are supported by the design of the platform - in order to provide a service. In other words, if a CPS administrator says something, PTT will believe it provided there is a guarantee that the former has *actually said it, what was said has not been altered, and has been said recently.*

Within the first approach, it is possible to distinguish two cases. The initiating end may authenticate its users without any other support. On the other hand, it may make use a trusted third party, which is shared with other peer machines. This choice has implications for trust, but also having all possible users of CNM registered on a single trusted system would make administration more easy to manage. Such a third party machine may also be used to allow machines - X and Y, for example - to authenticate each other.

Authentication Information

This is the information used to prove identity of users and machines which includes passwords and keys. These may be used in conjunction with other technology such as smart cards. The information needs to be protected from modification, and depending on the kind of cryptographic quantity involved, in addition divulgence.

Authentication Protocols

Depending on the choice of authentication mechanisms - public key, symmetric key - appropriate authentication protocols can be provided. Several protocols have been developed using both public key and symmetric technologies [6], [8], [9], [7], [10]. There is no need to reinvent another protocol from scratch. We will be describing the protocol details in another follow-up paper.

As part of this authentication process, a common conversation key can also be exchanged between initiator and target, which can be subsequently used to protect communications between them.

Another aspect to consider is the nature of communications between X and Y. If the communication is such that the two machines set up a link which would exist for a session allowing exchange of several messages, then the mutual authentication can be performed once per session. An alternative option is the provision of message by message authentication.

4.2 Against Unauthorized Access

To counteract the threat of unauthorized access, we require access control mechanisms. In providing such mechanisms, we need to consider what access control information is used in the decision making process, including access rules, and what entities – human or software – are responsible for supplying or modifying such information, inferring access decisions, and enforcing those decisions. Naturally, these objects need to be supported by the authentication service.

Consider again the case where a user U through CNM application A in machine X requests for information stored in a database through application B in machine Y. Once again, B in our case is the target DRS. We will assume that the authentication of the user to his local machine, and between machines, has taken place. That is, DRS in Y knows that the request is from U/X.

Enforcement

As mentioned earlier in Section 3.2, DRS can now enforce controls to decide whether to grant the request or not. It can check the role of user U (administrator, owner, user) of the partition in which the requested data object resides, and depending upon the role, the corresponding rights the user has (for operations) and whether they match with the operation being requested. As such, the target DRS is the final controller in this process.

Access Control Information

Access to organisation private data would be limited to users belonging to that organisation. TO provided data is intended to be shared between the TO and the appropriate customer. The information contributing to the access control decision include :

- Decomposition into partitions.
- User to role mapping (which user has what role with respect to which partition).
- User to group mapping, and group to rights mapping.
- Rights allowed by each role (administrator, owner, user)
- Simple access rights: read, write, create, delete or more generally method names
- Rights to grant and revoke simple access rights

4.3 Against Communication Security Threats

Integrity of information transferred between X and Y can be provided using an integrity cryptographic checksum. Calculation of such a cryptographic checksum involves the use of

a key. The choice of the key depends on the cryptographic system and the security protocols used. For instance, this key could be either the conversation key which was established as part of the mutual (X-Y) authentication process described earlier, or the private key of a public key system.

Confidentiality of information transferred between X and Y can be provided using encryption mechanisms. This once again, involves the use of a key. This key could be the conversation key which was established as part of the mutual (X-Y) authentication process described earlier.

4.4 Auditing

Finally, there could be an additional security requirement for audit logs. This can be regarded as complementary to access control enforcement.

- Logging accesses to database information (VAI): who (userid) from what machine (machine id) did what operation (operation id) on what object (objectid) at what time (timestamp), and the result of the request (granted or not)
- Allowing the administrator to define the type of events which should be logged, and the severity status.

4.5 Architectural Components

Authentication Components

We have a single Authentication/Certification Server principal in a domain, and an Authentication Service Component principal on each machine. The role played by the Server is dependent on the type of cryptographic technology used - public key or symmetric key. For instance, here is one possible scheme : The Server acts as a Certification Server and it retains public keys and certificates associated with the principals, the principals being the users, applications and machines.

When a principal in machine X (e.g. an application A, X_A) wishes to request a service from another principal on a remote machine Y (e.g. DRS), their respective Authentication Service Components communicate. If the Authentication Service Component of machine X is not aware of its counterpart component of machine Y, then it will make use of the Certification Server as a directory to obtain the certificate containing the public key of Y. Now mutual authentication between X and Y can occur and a common conversation key between X_A and DRS can be established.

Access Control Components

In this environment, we can have the access control component residing within or as part of the DRS. This access control decision information needs to be accessible to DRS. There also must be interfaces for initializing and updating this information. We can have the following interfaces : (a) An Administration Interface that allows administrators and policy setters to specify the policy information. (b) An Evaluation Interface that allows the DRS application to evaluate the request at runtime. There may be an additional interface to support for policy auditing purposes.

Communication Security Components

These include cryptographic modules providing encryption mechanisms (symmetric key and public key algorithms), hash functions, and providing secure storage.

Auditing Components

In this environment, we can have the auditing component residing within or as part of the DRS to log accesses to database information. There must be interfaces for specifying the type of events which should be logged and how the logs can be queried and analysed, and for enabling logging to occur at runtime.

5 Possible Security Solution in Stages

Following this analysis of possible security threats and requirements in the CNM environment, we can consider security enhancements to be carried out in stages. At the first stage, we might consider the following (See Figure 2) :

5.1 User Authentication Mechanisms

For each CNM machine, there will be a local component of the authentication service (AS). This will carry out authentication of local users, providing information that will serve as proof of identity to the target DRS in any management activity during that user session. A target DRS will in turn rely on its local AS component to validate the authentication proof presented to it. Users and AS components will be registered at a single certification server (CS) on a trusted machine.

- Authentication Information
 - User : user id, password, key
 - Machine (AS component): machine id, key
- Local component of the Authentication Service (AS) in each system
- Protection of authentication information during transfer
 - Public key based mutual authentication protocols between machines
- Establishment of security parameters between initiating application and target (DRS) providing the basis for making access control decisions.
- Registration of CNM users and AS components at a single Certification Server

5.2 Access Control Mechanisms

- Access control will be based on user id, machine id, and user role.
- Policy will reside at the target DRS. Role information will enable management rights at the customer site to be represented, whereby some restricted set of users at the customer site can be granted the ability to change the access privileges of the other valid users, and to register new valid users.
- Access control enforcement done by the target DRS, deciding whether to grant access or not using the authenticated information (from the request), and the access control policy information at the target.

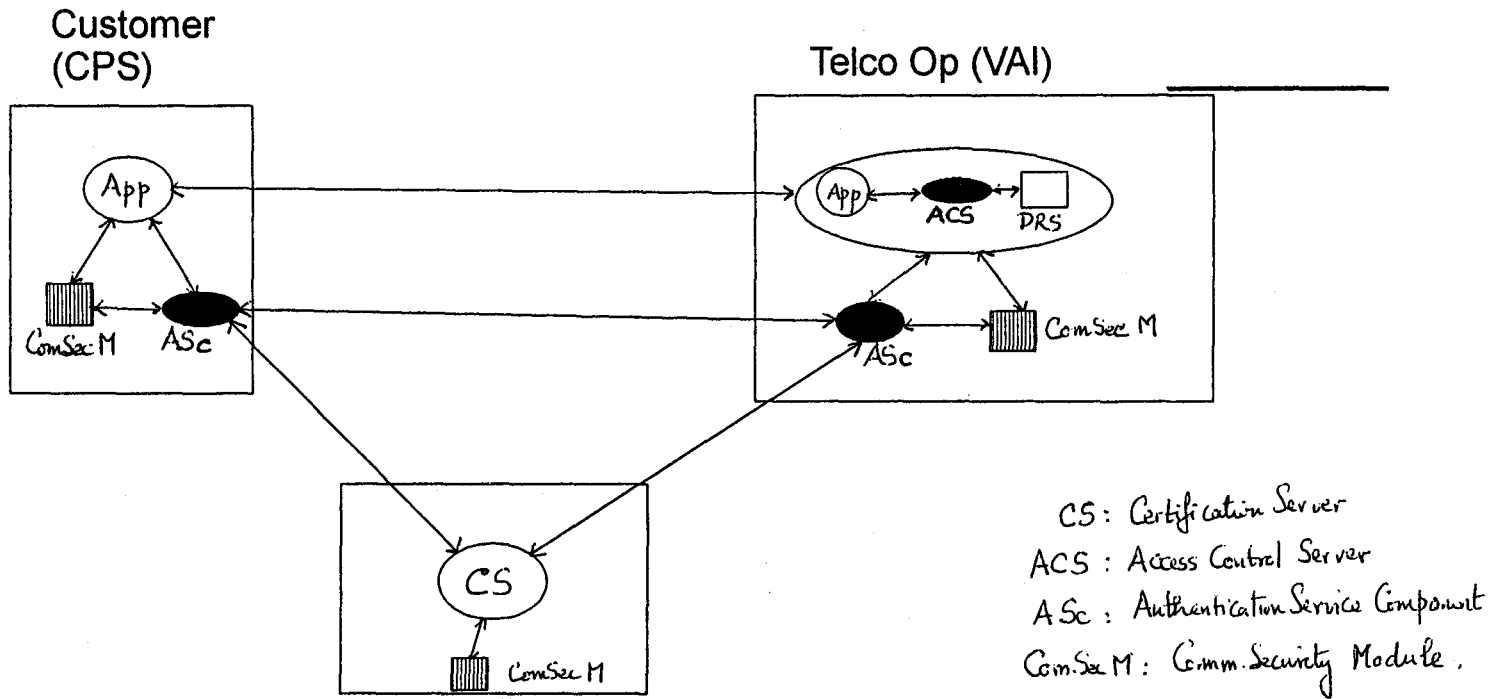


Figure 2 : Security in CNM : Architecture Components - Stage 1

5.3 Remaining Issues

The chief remaining issue is the selection of the cryptographic techniques to be used to provide the authentication service. The choice is between a using a pure asymmetric key approach and using a mixed symmetric and asymmetric approach [5]. In the latter case, asymmetric techniques are used for long-lived security information, while symmetric techniques are used for message by message protection within the scope of a user session, or the binding between a management application and the target DRS it accesses.

In determining which choice is more appropriate for the CNM environment, here are some of the factors contributing to the decision.

5.3.1 Trust and Performance

In the mixed scheme, following the initial authentication between the ASs on the initiating machine and the target machine where DRS resides, a conversation symmetric key is established. This symmetric conversation key is used within the scope of a binding between a management application and the DRS. If this key were compromised within a machine accidentally, the consequences would be very limited. In that case, it is reasonable to give such keys to the initiator (management application) and target DRS for their direct use. Message by message protection (authenticity and integrity) is achieved without a context switch.

In a pure asymmetric scheme, assuming a single level of keys, the private key component is involved in providing the authenticity and the integrity of requests between the management application and the DRS. The consequences of compromise of this private key would be more serious than in the case of the mixed approach. This leaves the choice of investing greater trust in the construction of the management applications and DRS or retaining the keys in a separate address space, incurring a performance penalty for message by message protection. Furthermore computation based on an asymmetric approach is more intensive than a corresponding symmetric one.

5.3.2 Confidentiality

Symmetric techniques offer much better performance for the provision of confidentiality. If confidentiality is to become necessary, it is better to choose the mixed scheme.

5.3.3 Availability of the technology

The best asymmetric algorithm to use (on technical grounds alone) remains RSA [3]. Within the USA, its use involves obtaining licenses from RSA Inc. There are several symmetric algorithms, the popular one being DES [4]; however use of such algorithms for confidentiality purposes is subject to export regulations.

Summary

In this paper, we have analysed the security requirements for CNM applications, and considered some of the design choices in the provision of appropriate security services, and the security architectural components that are required. The trend is that CNM is increasingly becoming important in the telecommunications market, where there is a greater need for

giving customers access to information such as how much they are being charged and for what services.

References

- [1] Bellcore, *SMDS Customer Network Management Service*, Technical Advisory TA-TSV-001062, Issue 2, Feb.1992.
- [2] ISO 7498-2, "*Information Processing Systems - Open Systems Interconnection - Reference Model - Part 2 : Security Architecture*", ISO 1988.
- [3] R.Rivest, L.Shamir, L.Adleman, "*A method for obtaining digital signatures and public key cryptosystems*", Commun. ACM, Vol.21, No.2, 1978, pp120-126.
- [4] National Bureau of Standards, "*Data Encryption Standard*", FIPS Pub.46, NBS, US Dept. of Commerce, 1977.
- [5] G.J.Simmons, "*Symmetric and Asymmetric Encryption*", Computing Surveys, Vol.11, No.4, 1979.
- [6] John Kohl and B.Clifford Neuman, *The Kerberos network Authentication Service*, V5, Draft 4, Dec.1990.
- [7] Open Software Foundation, *Distributed Computing Environment (DCE) Rev.1.0 - Introduction : Chapter 3.5*.
- [8] International Organization for Standardization (ISO), ISO/IEC JTC1/SC27 : Peer Entity Authentication Exchange Mechanisms
- [9] International Organization for Standardization (ISO), ISO/IEC JTC1/SC27 : Key Management : I, II and III.
- [10] Vijay Varadharajan, Phillip Allen and Stewart Black, *An Analysis of the Proxy Problem in Distributed Systems*, Proc. of the 1991 IEEE Symposium on Research in Security and Privacy, 1991.

Acknowledgements

The author would like to thank anonymous referees and his colleagues Phillip Allen and Jonathan Griffin for their valuable inputs.

SUPPORT FOR SECURITY IN DISTRIBUTED SYSTEMS USING MESSIAHS

Steve J. Chapin
Department of Mathematics
and Computer Science
Kent State University
Kent, OH 44242-0001
sjc@cs.kent.edu

Eugene H. Spafford
COAST Laboratory
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1398
spaf@cs.purdue.edu

Abstract

The MESSIAHS project is investigating the construction of a set of mechanisms to support task placement in autonomous, heterogeneous, distributed systems. In this paper we explore aspects of the MESSIAHS system that support security in distributed systems.

In particular, we will concentrate on aspects of MESSIAHS that defeat denial of service attacks, provide firewalls, protect private system description information, and support matching of tasks and systems based on security ratings. Development of these features will allow tasks to be scheduled in a heterogeneous distributed system, while protecting data and system integrity.

MESSIAHS is a set of mechanisms that ties together disparate computing resources to achieve distributed processing without sacrificing local control. MESSIAHS is novel in that it includes support for autonomous systems while providing flexible, scalable mechanisms to implement scheduling algorithms for heterogeneous distributed systems.

Keywords: distributed systems, scheduling, security, autonomy, availability, visibility

1 Introduction

We are investigating scheduling support mechanisms for autonomous, heterogeneous, distributed systems. Our goal is to develop mechanisms that allow scheduling algorithms to be implemented for large-scale distributed systems using heterogeneous hardware and software, across administrative boundaries. Such large-scale distributed systems can achieve performance surpassing that of the largest parallel supercomputers [11], and increase utilization of underutilized computing power [8]. As part of

this work, we have developed a set of mechanisms and a prototype implementation called MESSIAHS: Mechanisms Effecting Scheduling Support In Autonomous, Heterogeneous Systems [5, 4].

Our research is motivated by three factors. First, decentralization of computing systems has introduced administrative domains as a barrier to distributed computing. To overcome this, some method must be found to unite systems from incompatible administrative domains while respecting the autonomy of the individual systems. Second, many researchers have concentrated on scheduling and load-balancing algorithms while assuming the existence of the mechanisms necessary to support them (see, for example, Sarkar and Hennessy [13], Lo [12], or Blake [1]). They have either designed ad-hoc mechanisms to support particular algorithms, or limited their research to theoretical analysis of the scheduling algorithms. Third, users of computer systems may require resources that are not available locally, such as specialized processors or remote databases.

This paper concentrates on security aspects of the MESSIAHS mechanisms, which ties in with the first factor listed above. As part of the support for distributed computing across administrative domains, MESSIAHS provides mechanisms that

1. thwart denial of service attacks,
2. can act as a firewall to limit access by outside systems,
3. can restrict the flow of sensitive system description information outside an administrative domain,
4. and allows systems and tasks to be labeled in support of partitioning based on security requirements.

Section 2 gives background information describing the MESSIAHS system. Sections 3, 4, 5, and 6 describe the MESSIAHS mechanisms that support the four points listed above. Section 7 contains concluding remarks and proposes future directions for our investigation of security in distributed task placement.

2 MESSIAHS Background

Our systems are structured in a hierarchical fashion based on *virtual systems* representing administrative domains. A virtual system is composed of a set of subordinate virtual systems. Within each of these sets there can be many machines, which could be further grouped into virtual systems. At the lowest level, each machine is the sole member of a virtual system. We call an encapsulating virtual system a *parent*, and a subordinate system a *child*. Children with the same parent are called *siblings*.

For example, figure 1 displays part of the administrative structure of the Kent State University Mathematics and Computer Science Department. Within the department, there are several generally accessible machines such as Chaos and Nimitz, as well as machines supporting specialized research projects. One of these projects is

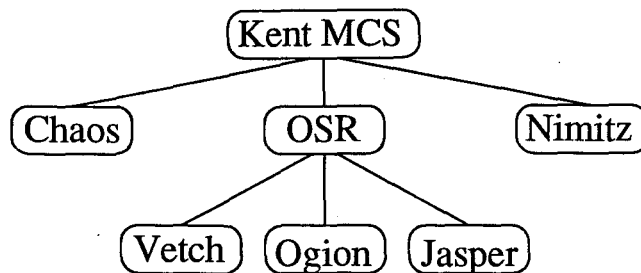


Figure 1: A subset of the machines in the Kent State Math/CS Dept.

the Operating Systems Research (OSR) project, which has administrative authority over a set of machines including Ogion, Vetch, and Jasper.

In MESSIAHS, each virtual system in the hierarchy has a scheduling support module that is responsible for maintaining the set of information required by the scheduling policy. Scheduling algorithms take a set of tasks and a description of the underlying multicomputer and devise an assignment of tasks to processors according to an optimizing criterion.

Our method for supporting scheduling decisions has three main parts: the *system description vector*, the *task description vector*, and the update protocol used to communicate between systems. The description vectors contain state description information, including system processing load, memory statistics, processing capabilities, and storage capacities. The update protocol sends system description vectors between modules.

The model for update flow is that a module collects several description vectors, adds information describing the local system, and condenses the resulting set of description vectors into one vector to facilitate scalability. This vector will be advertised to its parents and children. The module can also decide not to include data in the outgoing vector based on security constraints.

When a task is submitted for execution, a task description vector is sent to a scheduling module.¹ The scheduling module compares the task description to its own system description and the system description vectors it has received from other systems. Based on the scheduling policy, the module chooses one of the systems and attempts to schedule the task there.

MESSIAHS attempts to sacrifice the least autonomy for participating systems. There are four types of autonomy in distributed systems, as defined in [9, 6, 7], and refined in [4]: *execution autonomy*, *communication autonomy*, *design autonomy*, and *administrative autonomy*. Execution autonomy means that each system decides whether it will honor a request to execute a task; each system also has the right to revoke a task that it had previously accepted. Communication autonomy means that each system decides the content and frequency of state advertisements, and what other messages it sends. A system is not required to advertise all its capabilities, nor is it required to respond to messages from other systems. Design autonomy gives the architects of a system freedom to design and construct it without regard to existing

¹These requests are called *scheduling requests*.

systems, yielding heterogeneous systems.

Administrative autonomy means that each system can have its own usage policies and behavioral characteristics, independent of any others. In particular, a local system can run in a manner counterproductive to a global optimum. In the usual case, scheduling modules will cooperate, but administrators must be free to set their local policies or they are unlikely to participate in the distributed system [2, 8].

The next four sections examine the behavior of the module and show how the autonomy support within MESSIAHS facilitates security in distributed systems.

3 Denial of Service Attacks

Communication, administrative, and execution autonomy form a basis to thwart denial of service attacks. Each system can autonomously decide whether or not to accept any task. Thus, policies can be written to use current load or the identity of a requesting system as criteria to screen incoming requests.

MESSIAHS implements two interface layers that scheduler-writers can use to implement their algorithms. The first, called the *MESSIAHS toolkit*, is a library of function calls that can be used with a high-level language such as C [3]. The second, the *MESSIAHS Interface Language*, or MIL, is an interpreted language that is especially tailored to the task of scheduling [5].

Either of these interface layers can be used to implement scheduling *filters*. A filter takes two description vectors and returns a numerical result indicating how well they match. A *task filter* compares an incoming task description vector to a system description vector and returns an integer. A negative number indicates an error during the evaluation of the filter, while zero indicates that there is no match. In either case, the task is not accepted for input. Positive integers indicate a match. In general, larger values imply a better match, although a boolean filter can be implementing by returning the same value for all matches, e.g. the integer one.

For example, the local policy could decline scheduling requests when the local load average exceeds a threshold. This would limit the impact of outside tasks on the system, although it would not discriminate between legitimate and malicious requests for resources. A policy based on the source of the request could ensure that the task comes from a trusted source. A mixture of these policies could limit the number of tasks from untrusted sources while also limiting the total load on the system. In this way, an attempted denial of service will consume at most a small percentage of the resources of the machine.

Communication autonomy can also help to defeat denial of service attacks. Because a system is not required to respond to a message, it can simply ignore suspicious scheduling requests. This diminishes the possibility of saturating the scheduling module with requests from outlaw systems. It also eliminates a possible covert channel, wherein an attacker could study the behavior of the system in response to spurious scheduling requests.

In addition, execution autonomy allows the scheduling policy to revoke or migrate running jobs. This facility can be used to remove tasks consuming excess resources,

or to respond to a surge in load caused by an attempted denial of service attack.

4 Firewalls

It is sometimes desirable to mask the details of a resource, while still allowing outside access. This is commonly done for electronic mail systems, and is usually implemented through the use of a *firewall* [10]. All attempts to access a resource pass through the firewall, and the outside agent accessing the resource cannot tell the exact location of the resource.

For example, in figure 1, the OSR node can act as a firewall to hide the presence of Vetch, Ogion, and Jasper. It can still advertise some of their capabilities to the other nodes in the system, but it appears as if all their resources are located at the OSR node.

MESSIAHS incorporates two mechanisms to accomplish this: information condensation and proxy acceptance. Information condensation takes place when two or more update vectors are combined to form a single vector for advertisement. For example, OSR combines the capabilities of OSR, Vetch, Ogion, and Jasper into a single vector that can be sent to nodes outside the virtual system rooted at OSR. In the process, all identifying information, such as location information of individual resources, is removed.

For example, suppose Ogion were an SGI Indy running IRIX 5.1², Vetch were a SPARC IPC running SunOS 4.1³, and Jasper were a 486 clone running FreeBSD. The information advertised by OSR would indicate the presence of MIPS, 486, and SPARC processors, as well as the presence of the IRIX, BSD, SunOS operating systems. There is no indication which processor is running which operating system. The OSR node knows this, but does not advertise it to the outside world. This might cause another node to send a spurious request to OSR, e.g. a request to run a task on a SPARC processor running the BSD operating system. However, OSR will have enough information to discard the request, and no tasks will be misscheduled as a result.

This leaves open the question, "If a task is scheduled on a system, how is it moved to the system without the originator knowing where the system is?" The solution used in MESSIAHS is the *proxy accept*. When passing a scheduling request to an interior node, the firewall logs the request, replaces the originator's address with its own address, and waits for the response from the interior node. If the node accepts the request, it sends an acceptance message back to the firewall.

Upon receipt of the accept message, the firewall replaces the address of the acceptor with its address, and forwards the acceptance to the originator of the request. The originator then treats the firewall as the acceptor, and forwards the task for execution. The firewall then forwards the task to the real acceptor, and continues to act as an intermediary between the acceptor and the outside world.

²Indy and IRIX are trademarks of Silicon Graphics, Incorporated.

³SPARC and SunOS are trademarks of Sun Microsystems, Inc.

```

struct statvec {
    float min, max, mean, stddev, total;
};

typedef struct statvec Statvec;

struct procclass {
    bit32    nsys;        /* number of machines in this class */
    Statvec  qlen;       /* run queue statistics */
    Statvec  busy;       /* load on cpu (percentage) */
    Statvec  physmem;    /* total physical memory */
    Statvec  freemem;    /* available memory */
    Statvec  specint92;  /* ratings for specint 92 */
    Statvec  specfp92;  /* ratings for specfp 92 */
    Statvec  freedisk;   /* public disk space statistics */
};

```

Figure 2: Statistics vectors and processor classes in MESSIAHS

5 Control of Advertised Information

To be secure, systems must not advertise sensitive information to untrusted systems. The communication autonomy support in MESSIAHS allows scheduling policies to omit data from their outgoing vectors. This feature can be used to filter outgoing data to be consistent with a security policy.

To facilitate scalability, MESSIAHS uses a statistical representation of the capabilities of a virtual system. That is, instead of listing specific ratings of individual machines, the minimum, maximum, mean and standard deviation for a capability are kept, as well as the number of systems represented in a vector (see figure 2).

To partition the possible space of attributes, machines are divided into classes based on logarithmic scale of their processor speed, with a structure containing statistical information regarding the available resources for machines in each class (see figure 2). In this way, information can be condensed while still providing enough information for scheduling algorithms to make intelligent choices.

MESSIAHS provides routines to automatically combine multiple statistical vectors into one. This is the mechanism used by the module to coalesce multiple system descriptions into the description of a single virtual system. The autonomy support within the mechanisms allows fields to be omitted from the combination. For example, if the OSR project administrator does not want the capabilities of Jasper advertised to nodes outside the project, he can specify that Jasper's resources not be included in OSR's advertised vector. Both MIL and the scheduling toolkit allow the administrator to restrict information advertisement in this fashion.

```
begin combining
  string $out.tier      not match($out.tier, "preferred"):
                        set $out.tier + ":preferred";
  string $out.department not match($out.department, "research"):
                        set $out.department + ":research";
end
```

Figure 3: A code fragment from MIL using labels

The obvious tradeoff in this scheme is the size of the advertised vectors versus the degree of detail present in the vectors. The approach taken allows the vectors to be kept to a reasonable size⁴ while still providing sufficient visibility of individual machines so that scheduling algorithms can function well.

6 Extension and Labeling Support

In addition to the fixed data represented by statistical vectors, MESSIAHS also allows administrators to extend the system description vector. This affords the mechanisms flexibility in supporting scheduling algorithms, and can be used to support secure processing based on security classifications.

Systems can insert labels in their extension vectors to indicate the security classification required to run a task on that system. Tasks can include a security label listing their security classification. The scheduling algorithm can match the levels to ensure that the task's security rating is equal to or higher than that of the system.

The MESSIAHS mechanisms can improve the efficiency of a distributed computation. Large jobs can be partitioned into smaller tasks based on their security requirements, and then only those tasks that require secure processing will be run on secure sites. Tasks that do not require secure processing can be run on any general-purpose processor within the distributed system. This not only reduces the load on the secure installations, it increases the security of these systems by ensuring that only computations that require secure resources are run there.

This labeling mechanism could also be used in commercial systems. Within a single organization, tasks could be labeled with their department of origin, e.g. sales or research. Systems could protect private data by only executing certain classes of jobs. This mechanism could also be used by an institution that sells processing time to outside customers. The institution could offer different tiers of service, and jobs from customers would be labeled based on the tier they had purchased. Jobs from more expensive tiers might receive preferential treatment by being given higher priority, or being assigned to faster computers.

Figure 3 shows an example usage of labeling written in MIL. Assume that the

⁴The update vectors in the prototype implementation are approximately two kilobytes in size.

intent is to advertise that the virtual system will run tasks for preferred customers within the research department. This code fragment makes sure that the service tier preferred appears in the outgoing description vector, and ensures that the research department label also appears. Again, outside systems cannot determine if the preferred tier applies to the research department, but this will not cause a breach of security.

Two factors complicate the use of MESSIAHS for this type of service. First, there must be some method of ensuring that machines and tasks cannot spoof higher security classifications or service tiers. Second, there must be guarantees that the data in the extension area remains private and uncorrupted, because communication autonomy allows intervening systems to read or alter the contents of an advertisement. In the absence of a distributed secure network, we are left to devise software solutions to these problems.

We can use well-known authentication techniques such as message digests, digital signatures, and public-key encryption to ensure the validity of labels and vector contents (see, e.g. [14]). A possible solution to the second problem is to encrypt private data within the extended portion of the task description vector so that only trusted hosts can view the secret data. However, this scheme presents the difficulty that intermediate nodes have no semantic knowledge of the encrypted information, and therefore cannot apply any combining rules to condense the information. Finding a clean solution to this dilemma is an open problem.

7 Concluding Remarks

We have described the MESSIAHS system for scheduling support. MESSIAHS includes generous support for autonomy in distributed systems, and this autonomy support can form the basis for security measures.

We have shown how the scheduling support mechanisms can support four aspects of security: thwarting denial of service attacks, acting as a firewall, restricting the flow of information outside an administrative domain, and allowing systems and tasks to be matched based on their security requirements.

The MESSIAHS system has several potential applications for distributed systems in a trusted environment. The mechanisms can support process migration and load balancing. Because the update protocols track which machines are available, fault tolerance can be layered over the mechanisms. The revocation facility can support transaction management in a nested-transaction environment.

Our plans for the future are to study the issue of cryptographic techniques to handle end-to-end security issues. However, there are significant barriers to be overcome to prevent nodes from advertising encrypted, sensitive information outside an administrative domain.

References

- [1] B. A. Blake. Assignment of Independent Tasks to Minimize Completion Time. *Software-Practice and Experience*, 22(9):723-734, September 1992.

- [2] A. Bricker, M. Litzkow, and M. Livny. Condor Technical Summary. Technical Report 1069, Department of Computer Science, University of Wisconsin-Madison, January 1992.
- [3] S. Chapin and E. Spafford. Implementing Scheduling Algorithms Using MESSIAHS. *Scientific Programming*, 1994. to appear in a special issue on Operating System Support for Massively Parallel Computer Architectures.
- [4] S. J. Chapin. Scheduling Support Mechanisms for Autonomous, Heterogeneous, Distributed Systems. Ph.D. Dissertation, Purdue University, 1993.
- [5] S. J. Chapin and E. H. Spafford. Constructing Distributed Schedulers with the MESSIAHS Interface Language. In *27th Hawaii International Conference on Systems Sciences*, volume 2, pages 425–434, Maui, Hawaii, January 1994.
- [6] W. Du, A. K. Elmagarmid, Y. Leu, and S. D. Ostermann. Effects of Local Autonomy on Global Concurrency Control in Heterogeneous Distributed Database Systems. In *Second International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, pages 113–120. IEEE, 1989.
- [7] F. Eliassen and J. Veijalainen. Language Support for Multidatabase Transactions in a Cooperative, Autonomous Environment. In *TENCON '87*, pages 277–281, Seoul, 1987. IEEE Regional Conference.
- [8] C. A. Gantz, R. D. Silverman, and S. J. Stuart. A Distributed Batching System for Parallel Processing. *Software-Practice and Experience*, 19, 1989.
- [9] H. Garcia-Molina and B. Kogan. Node Autonomy in Distributed Systems. In *ACM International Symposium on Databases in Parallel and Distributed Systems*, pages 158–166, Austin, TX, December 1988.
- [10] S. Garfinkel and E. Spafford. *Practical UNIX Security*. O'Reilly and Associates, 1991. ISBN 0-937175-72-2.
- [11] A. H. Karp, K. Miura, and H. Simon. 1992 Gordon Bell Prize Winners. *IEEE Computer*, 26(1):77–82, January 1993.
- [12] V. M. Lo. Task Assignment to Minimize Completion Time. In *Distributed Computing Systems*, pages 329–336. IEEE, 1985.
- [13] V. Sarkar and J. Hennessy. Partitioning Parallel Programs for Macro-Dataflow. In *ACM Conference on Lisp and Functional Programming*, pages 202–211, August 1986.
- [14] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1994.

A TECHNICAL APPROACH FOR DETERMINING THE IMPORTANCE OF INFORMATION IN COMPUTERIZED ALARM SYSTEMS*

David S. Fortney
Lawrence Livermore National Laboratory
P.O. Box 808, L-195
Livermore, CA 94550 USA
dfortney@llnl.gov

Judy J. Lim
Lim & Orzechowski Associates
198 Stone Valley Way
Alamo, CA 94507 USA
jlim@sandia.gov

ABSTRACT

Computerized alarm and access control systems must be treated as special entities rather than as generic automated information systems. This distinction arises due to the real-time control and monitoring functions performed by these systems at classified facilities and the degree of centralization of a site's safeguards system information in the associated databases. As an added requirement for these systems, Department of Energy (DOE) safeguards and security classification policy is to protect information whose dissemination has the potential for significantly increasing the probability of successful adversary action against the facility, or lowering adversary resources needed for a successful attack. Thus at issue is just how valuable would specific alarm system information be to an adversary with a higher order objective. We have developed and applied a technical approach for determining the importance of information contained in computerized alarm and access control systems. The methodology is based on vulnerability assessment rather than blanket classification rules. This methodology uses a system architecture diagram to guide the analysis and to develop adversary defeat methods for each node and link. These defeat methods are evaluated with respect to required adversary resources, technical difficulty, and detection capability. Then they are incorporated into site vulnerability assessments to determine the significance of alarm system information in the context of a facility attack. This methodology was successfully applied to the Argus alarm, access control, and assessment system developed at the Lawrence Livermore National Laboratory. Argus is software-driven, contains interrelated databases, shares host computers, and communicates with field processors and alarms through a common network. The evaluation results provided insights into the importance of alarm system information while the methodology itself provided a framework for addressing associated information protection issues.

INTRODUCTION

Computerized alarm and access control systems are increasingly being used at classified facilities to automate site security by providing real-time control and monitoring. Given the functions they perform, these systems and their associated databases are rather unique entities and should not be viewed just as automated information systems. Adding to the complexity of the situation is the DOE safeguards and security classification policy [1] to protect information whose dissemination has the potential for providing adversaries with significant advantages for a successful attack. Thus recently DOE has attempted to address the protections that must be provided to computerized alarm systems in the new classified computer security order. The key issue lies in the determination of the importance to an adversary of the information being processed or transmitted in the alarm system. This paper presents a technical approach that is

*Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

based on the discipline of vulnerability assessment to address this issue. Prior to presenting the methodology, we will first discuss the function of an alarm and access control system and the role it plays in the overall site safeguards program. We also need to be aware that the information in the alarm system may not be classified, but that the system itself serves to protect other classified matter.

Alarm systems

Sites handling or storing sensitive information or matter have sensors to detect potential intrusions, systems to monitor these sensors and assess alarms, and entry control systems to allow routine access by authorized personnel. Recent sophisticated computer-controlled alarm and access control systems have supplanted previously-used electromechanical systems devoid of any centralized intelligence. Because of the computerization and integration of these sensors and the accompanying wealth of information processed and stored by these alarm systems, computer security and information protection requirements now apply to these "intelligent" systems in addition to many regulations specific to alarms. Computerization provides a new medium by which safeguards system components may be defeated, increases the speed in which the adversary can achieve the defeat, and can decrease the detectability of the compromise or defeat. Alarm system integration using computers introduces the potential for single points of failure in the site safeguards system; that is, the adversary may only need access to a database to defeat several diverse sensors. Lastly, the centralization of data in these alarm systems potential poses operations security concerns in that the accumulation of the various data could provide sufficient information to let an adversary circumvent site safeguards.

The LLNL-developed Argus system [2] is an example of a computer-controlled, state-of-the-art alarm system. Argus is comprised of three main subsystems: the alarm subsystem, the entry control subsystem, and the security console which provides a map-based alarm assessment and response force communications capability (see Fig. 1). These systems share relational databases, host computers and field processors; and communicate with field processors using a common encrypted communications network.

The system services facilities at LLNL's main site, as well as several remote locations via encrypted microwave communications. Except for alarms or other anomalous conditions, the system is designed to operate without security personnel involvement. This is accomplished by using badge readers, accompanied by personal identification numbers (PINs) and biometrics to authenticate individuals.

Vulnerability assessment framework

Although systems like Argus play a significant role in a site's physical protection posture, they must be evaluated in the context of the overall safeguards and security system protecting the site against adversary missions such as theft or sabotage of sensitive information or other valuable assets. For an adversary to successfully attack a facility, he or she may have to defeat physical barriers, locks, human surveillance, and armed guards. Defeat of the alarm and access control system may or may not be an intermediate step for the adversary to accomplish the mission. The alarm system and its associated databases are only one part of the site safeguards system. Therefore, the vulnerability of the alarm system and the value of alarm system information can only be evaluated within the framework of a higher-order adversary mission that a site's safeguards and security system is designed to protect against.

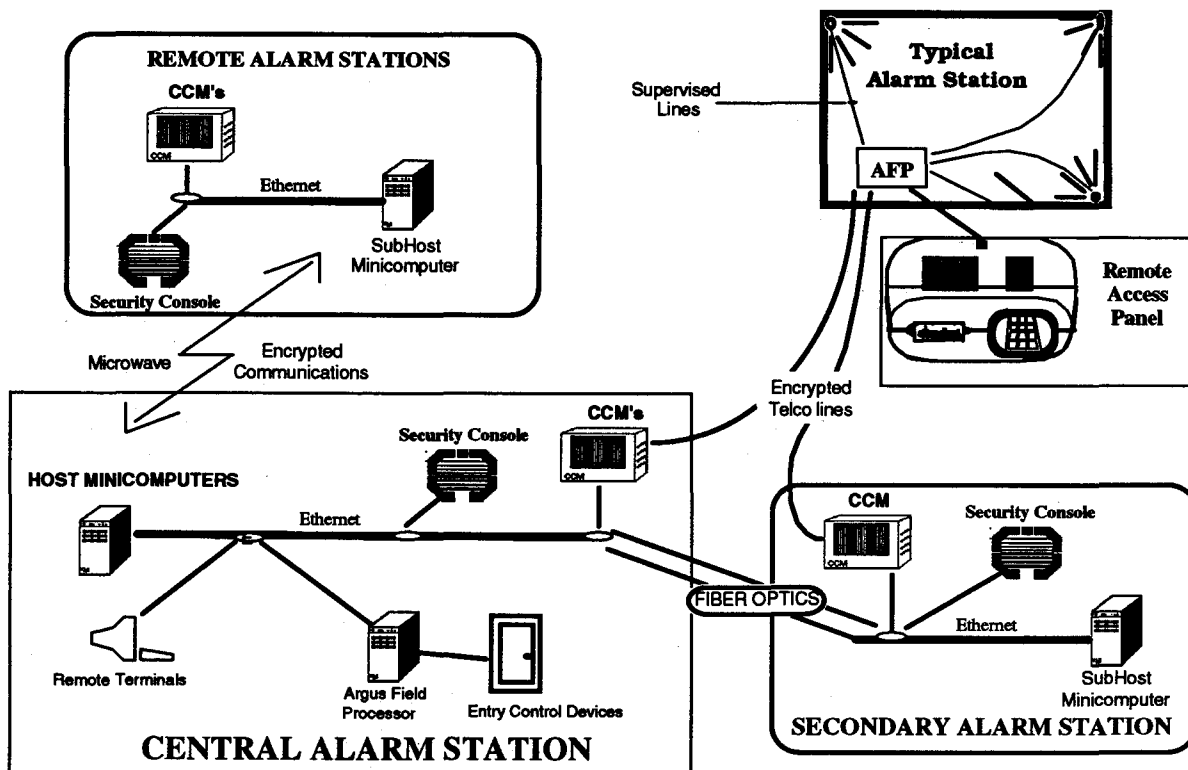


Figure 1. Argus system schematic

The framework for evaluation is provided by a site vulnerability analysis (VA) that identifies ways in which the site is open to attack or damage by a spectrum of threats. A site VA focuses on human adversarial threats, including insiders, outsiders, and combinations of insiders and outsiders possessing a range of capabilities, resources, and knowledge. These adversaries may attempt malevolent acts such as theft or diversion of sensitive material, radiological or toxicological sabotage, or compromise of classified information. The consequences of these acts can adversely impact public health and safety, national security, or lead to monetary loss. The site VA delineates the various scenarios that may be used by an adversary to attack a facility, evaluates the relative likelihoods of the scenarios being successful, and quantifies the baseline risk for the facility.

Facility assets are typically protected in a graded fashion, with the most valuable assets located within several concentric facility protection layers. An adversary must defeat and traverse each of these layers in order to reach the critical assets. The adversary will attempt to defeat each layer without being detected or unduly delayed. There may be many "defeat methods" the adversary could employ to defeat the layer. VAs usually assume that adversaries are aware of available options and make choices among these options to develop an optimal overall strategy that maximize their chances of success. Adversary success is a function of detection, assessment, interruption, delay, and neutralization. For example, it is customary to assume a nonviolent insider threat would choose to minimize the probability of being detected; a violent outsider force would be more concerned with minimizing delay so as to complete their mission before they can be interrupted (and neutralized) by the facility response force.

In conducting a VA, analysts need to clearly define the capabilities and the decision criteria of the threats under consideration. For each layer of the safeguards system, the analyst needs to think like the adversary to brainstorm and refine potential defeat methods. These defeat methods are then evaluated considering safeguards measures that would detect or delay the adversary. Using the adversary's decision criteria, an optimal overall strategy is determined. Computer models and tools have been developed to support the conduct of site VAs.

Our approach for evaluating the importance of alarm system information is to evaluate alarm system vulnerabilities in the context of a site VA. Exploitable alarm system vulnerabilities are identified, quantified, and included as part of an overall adversary strategy for perpetrating a higher-order goal—for example, theft of weapons grade plutonium. Critical alarm system information is then identified as that information that significantly increases the probability of adversary success or significantly reduces adversary resources needed for a successful attack.

EVALUATION METHODOLOGY

The methodology for determining the importance of alarm system information entails applying the VA process to a state-of-the-art system to determine potential vulnerabilities and their possible exploitation. These system vulnerabilities are folded into a site vulnerability assessment to determine if they provide the adversary with new defeat methods that are more powerful or more effective in accomplishing the overall objective. This methodology, developed to evaluate the LLNL-developed Argus system, entails six steps: (1) Threat definition; (2) Alarm system characterization; (3) Defeat method development; (4) Evaluation of defeat methods; (5) Incorporation into site VA; and (6) Determining importance of alarm system information.

Threat Definition

The first step in any VA is to define the relevant threats to be analyzed. Threats to be considered can include insiders, outsiders, and mixtures of insiders and outsiders, and threat parameters can include: the number of adversaries; equipment, skills, knowledge and financial resources; and objectives, motivation and willingness to use risky or violent strategies. For VAs of any computer-based system, it is especially important to define the level of technical knowledge and sophistication, and available resources for such activities as building specialized electronic devices and special-purpose computers that can cost millions of dollars. The DOE threat policy defines a spectrum of threats and threat capabilities to be considered in site VAs.

For our application, we consolidate the threats into three major threat types: outsiders and two types of insiders, privileged and nonprivileged. *Privileged insiders* are defined as those with special access and knowledge to alarm system information or systems, whereas *nonprivileged insiders* are other site personnel. Several subcategories of privileged insiders are considered to account for different levels and types of alarm system knowledge and access, and several subcategories of nonprivileged insiders are considered to account for different levels of access to our target of interest. From a classification perspective, the threats of interest are the nonprivileged insiders and outsiders, since the privileged insiders would likely have access to the classified information. However, from an information exploitation perspective, the privileged insider is of particular interest, since they are the most knowledgeable and could best attempt to misuse the information to exploit the alarm system.

Alarm system characterization

Computerized alarm and access control systems are comprised of central host computers or servers, field processors to monitor and report alarms and to control entry points, intermediate communications network and concentrators, computer terminals, and central and secondary alarm stations with alarm assessment and communications capabilities. These systems monitor and control various end devices, such as sensors, access control locks, badge readers, and closed circuit television cameras. In some systems, such as Argus, there are also remote field locations and secondary alarm stations with microwave communications links back to the central system.

Because of the interconnectivity of the alarm system and the number of types of communications links and end devices, a comprehensive VA must consider the possibility of a system attack from any point in the system. For instance, one cannot assume that a sensor could only be disabled at the sensor itself, or at a point between the sensor and the host computer. Depending on the configuration of the system, it may be possible to defeat a sensor via a totally separate path, such as from a seemingly unrelated communications link. To ensure a comprehensive and disciplined evaluation approach, a detailed system architecture diagram is developed. This schematic should show the architecture of the system, and delineate categories of devices (nodes) and communications connections (links) for analysis. At this stage, the analysts should gain an overview understanding of how the alarm system functions, and also understand the role of each of the nodes and links, as well as collect pertinent documentation and specifications.

Given this characterization of the physical configuration of an alarm system, the logical configuration implements a real-time control and monitoring of the sensors and the transmission, processing and protection of the associated data. Alarm system data consists of basic types of information: system configuration information such as sensor types and location, component specifications such as detection sensitivity levels and model types, personnel area access authorization information, personnel authentication information such as "protected" PINs and biometric parameters, and operational trends of the safeguards system in terms of the reliability of components and scheduled states of inactivity. For an adversary to access any of this information or use the alarm system, he or she must first defeat the system.

Defeat method development

As opposed to their simpler predecessors, computer-based alarm and access control systems are subject to a wide range of technically sophisticated defeat methods highly dependent on system design and implementation. To generate defeat methods for a particular system requires iterative interviews with system designers, hardware and software engineers, cryptologists and security personnel to understand the technical and procedural details of how the various components and communications links operate and could be exploited. Often the defeat methods are involved and require a series of adversary actions spread over time to defeat even a single communications link. The elicitation and defeat method development process involves an initial interview with the cognizant expert to gain a system understanding, obtain pertinent written documentation, and outline generic defeat possibilities. We then develop potential defeat methods for each node and link on the alarm system architecture diagram. In subsequent interviews of the same expert, we present the defeat methods and solicit feedback: Are the defeat methods plausible? If not, why (i.e., what would technically prohibit the defeat method)? Are the defeat method's implementation details correct? What is the easiest way for an adversary to implement the defeat

method? For certain complicated defeat methods, multiple experts have to be consulted and four or five interview iterations are necessary.

The defeat methods generated are highly dependent on system design and implementation. Some illustrative defeat method categories are:

- Sensor defeats or tampering
- Introduction or substitution of surrogate devices or components
- Communications link defeats such as tampering inside conduits or junction boxes; defeating polling or line supervision; installing intelligent eavesdropping devices or sniffers; and defeating line encryption by obtaining the encryption keys, breaking the cryptoalgorithm, or obtaining access to an unencrypted segment of the communications link [3,4];
- Capture of a privileged login to issue commands to tamper with alarm information
- Badge, PIN, or biometric device defeats
- Tampering with backup archives of system software or configuration data
- Tampering with source code (e.g., "trapdoor") or data prior to their being used on the production alarm system

Evaluation of defeat methods

Once the defeat methods have been generated and described in sufficient detail, they are evaluated in terms of their usefulness to the adversary in perpetrating the overall mission. For each defeat method, we consider three resource attributes: the amounts and types of resources required, the type and level of adversary knowledge and sophistication, and the amount of required access. Next we identify any safeguards that protect against each defeat method, and we quantify the likelihood of the adversary successfully completing the defeat method. We describe these steps in the following paragraphs.

The resource types include number of persons; type and cost of required equipment, including tools, computers, and electronic devices; and adversary task time. For some defeat methods test devices or surrogate devices may need to be constructed and tested ahead of time. When specialized devices or computers are required, it is important to ascertain their commercial availability. In estimating the required time for a defeat method, both the time to prepare away from the facility (including building test or surrogate devices, etc.) and the task time at the facility (including time to implant eavesdropping devices, etc.) should be included.

The adversary knowledge and sophistication levels include the type of expertise required (software, database, system programming, network communications, electronic components, etc.) and an estimate of the difficulty of the task. A constructed scale proves useful in delineating the inherent difficulty of the tasks as well as to differentiate task difficulty for the various threat types. For our application, we define the following scale:

Scale	Description of adversary skill level
Very Low	Requires no particular computer-related or technical knowledge.
Low	Requires competent adversary who knows general functional information about the system.
Medium	Requires competent adversary with detailed information about the system and/or specific computer knowledge and expertise.
High	Requires extremely knowledgeable and experienced adversary with extensive information about the system and/or extensive computer knowledge and expertise.
Very High	A difficulty level that may be theoretically possible, but in practice, even a team of experienced system programmers would not be assured of successful implementation.

Now we specify the required access to perpetrate the defeat method, such as access to physical areas of the facility, to privileged keys or combinations, to privileged computer accounts, or to privileged data. For those adversaries without the requisite access, we specify how that adversary may attempt to gain that access. This may involve reviewing existing site VAs, or performing partial VAs. We need to also consider the possibility that one alarm system defeat method could be used to gain the access required for another alarm system defeat method—for instance, gaining privileged computer logins may be one step of another alarm system defeat method.

Once the defeat methods have been evaluated for resource requirements, adversary knowledge, and access, we may be able to exclude certain defeat methods for use by certain adversaries. Defeat methods requiring resource attributes incompatible with those assumed for the adversary are eliminated (for that adversary). For example, we preclude outsiders using strategies that involve repeated physical access to controlled facilities with a closely knit work group.

Finally we identify the safeguards capable of preventing or detecting the adversary to estimate the probability of detection for each adversary type. For each defeat method, we estimate the probability of *timely detection* (i.e., probability that the facility detects the adversary before it is “too late”) by eliciting opinions from technical experts—engineers, system programmers, operators, and administrators. It was sufficient for our purposes to use a constructed scale for that probability assignment [5], with each scale point representing a probability range:

Scale	Probability range	Probability description
Very Low	0.0–0.1	Adversary is “heavily favored” to succeed.
Low	0.1–0.25	Adversary has the advantage, but is not “heavily favored.”
Medium	0.25–0.75	Neither adversary nor system has the clear advantage; “toss up”
High	0.75–0.9	System has the advantage, but is not “heavily favored.”
Very High	0.9–1.0	System is “heavily favored” to detect and interrupt adversary before mission completion.

In addition to timely detection, *after-the-fact detection* also plays a significant role for computerized systems. Timely detection focuses on the ability of the facility to detect and interrupt the adversary before mission completion, whereas after-the-fact detection is the ability of the facility or system to detect that a defeat method was attempted in the past. The primary value of after-the-fact detection is to provide traceability of adversary events and assurance that no successful attack has been perpetrated.

Incorporation into site VA

We assume that a site VA has already identified the feasible adversary scenarios to achieve the ultimate mission (e.g., the theft of weapons grade plutonium). These scenarios typically involve direct defeat of the actual safeguards components and procedures. The site VA has also quantified the associated likelihoods and impacts of the scenarios, determined the adversary's optimal scenario, and quantified the overall risk of the adversary mission. Thus a baseline risk has been established for comparison with criteria for risk acceptance. Now by incorporating the alarm system defeat methods into the site VA, we can explore whether it is advantageous for the adversary to use alarm system defeat methods in lieu of the methods previously identified in the VA. If alarm defeats provide adversary advantage, we go on to determine the value of alarm system information in the context of overall site risk.

The VA methodology uses computer models to assimilate the alarm system defeat methods to generate new adversary strategies for facility attack and to select an optimal adversary strategy from this set to quantify the underlying risk from each threat. This is accomplished by adjusting the optimal mission strategies from the site VA to include alarm system defeat methods when they prove more beneficial to the adversary than the non-alarm system defeat methods considered in the baseline site VA.

Integrating the alarm system defeat methods into the site VA framework allows us to assess the impact of the use of alarm system defeat methods and adversary access to alarm system information on the overall facility risk. For instance, use of these methods may provide the adversary with more efficient or powerful means of accomplishing the facility attack.

If a new attack strategy involving use of the alarm system information increases the facility's perceived risk unacceptably, safeguards upgrades to reduce the risk should be considered. These upgrades may be analyzed to ensure cost-effective risk reduction. The safeguards may either be applied to the alarm system, or to another aspect of the safeguards system, since what is important is reduction of overall facility risk. Moreover, if use of the alarm system information or defeat of components would significantly improve the adversary chances of success or significantly reduce the required resources for his mission, then the information or system component merits robust protection.

Determining the importance of alarm system information

We are now in a position to evaluate the information in the alarm system and its associated documentation, and to identify the information that is significant from a vulnerability perspective. Recall that the site VA has established the risk equivalences between adversary scenarios employing different defeat methods. That is, a theft scenario involving adversary tampering of door position sensors, locks, and intrusion detectors would be equivalent to a scenario with adversary tampering of the alarm system database to gain facility access and

inactivate alarms if both scenarios have the same detection probability. The importance of alarm system information is now evaluated within this framework.

Alarm system information may be evaluated as one of three general types: alarm system design information, alarm system configuration data, and facility vulnerability information. Each of these requires a slightly different approach to determine information value:

Alarm system design information is technical information on alarm and access control system design such as design philosophy, system architecture, hardware components, algorithms for protecting and processing alarms, message protocols, and database schemas. Much of this information may reside in technical system documentation as well as be directly observable from the alarm system itself. The critical alarm system design information is that which was found to be significant to an adversary in perpetrating a defeat method that proves useful for the overall mission. If this information resides only in written technical documentation and not deducible from other sources, it should be treated as very sensitive. If however, the information can be directly observed or derived from the alarm system itself, an evaluation is performed on the difficulty of doing this. If obtaining the information from the system can be done expeditiously or must be done necessarily in conjunction with using the information to defeat the system, then the information need not be considered particularly sensitive. If this is not the case, the information protection should be commensurate with the incremental difficulty of accessing the alarm system to obtain the information prior to using it in the defeat method.

Alarm system configuration data is site-specific information particular to the implementation of the alarm and access control system. Examples include facility maps, location, types, and sensitivity settings of sensors and other field hardware, rules for alarm configuration and mode changes, personnel access authorization data and rules, personnel authentication data such as PINs and biometric data, and encryption keys. We note that particularly sensitive configuration data, such as encryption keys, may only be contained in protected hardware components of the system, and not generally accessible in configuration databases. Alarm system configuration data found to be significant to an adversary in perpetrating a defeat method that facilitates the overall mission is evaluated in the same manner as system design information, and the information protection is commensurate with the difficulty increment.

Facility vulnerability information includes information potentially revealing vulnerabilities in the facility's safeguards and security system. The information may be information from various sources that do not by themselves reveal vulnerabilities, except by being integrated and "fused" to provide a "bigger" picture. This information may include such things as the presence of unalarmed pathways, or the fact that the facility is temporarily in a more vulnerable state (such as some emergency). This information must be evaluated in the context of non-alarm system security measures that may mitigate the potential vulnerability. For instance, the potential vulnerability implied by the fact that alarms are undergoing maintenance must be evaluated while considering the security contingency measures.

In all three categories, the primary focus is that alarm information which significantly improves the adversary's overall prospects of success, or significantly lowers the resources required for a successful attack. Also of interest is that information which allows the defeat of the alarm and access control system, but whose impact on overall mission success is nominal. Specifically, information which significantly improves the adversary's prospects can be defined to be

information which allows for a significant decrease in detection probability or task time for the optimal strategy. We note that some information appearing to be significant may be "nice to have" in advance, but would not materially affect strategy success since the information is obtainable in "real time" by the adversary.

Protection of significant alarm system information can be complicated, since protection mechanisms for sensitive information do not protect against the privileged insider adversary and may not protect against other threats any more than is already being afforded by the alarm system itself. To mitigate the risk of adversary use of significant alarm system information against the facility may require technical system upgrades rather than conventional physical and administrative information protection mechanisms. Also, alarm system designers should consider information importance during system design, and information of high utility to an adversary should be excluded from the system whenever possible. If such information must be included, system design should preclude its easy exploitation (e.g., critical encryption keys should reside only in protected hardware).

SUMMARY

We have developed and applied a vulnerability-based methodology for the evaluation of the importance of alarm system information to an adversary. Our evaluation approach is premised on the existence of a site VA which has established the baseline risk for the site relative to higher-order adversary missions. The site VA provides the relative framework for evaluation of alarm system information. The usefulness or importance of this type of information to an adversary is based on determining equivalences to sets of sensors, components, and procedures that an adversary would otherwise have to defeat in achieving his mission. Our application to a computerized, state-of-the-art alarm and access control system—LLNL's Argus system—have given us certain insights on the versatility of the evaluation approach and also particularly on the value of alarm system information to an adversary. Although this methodology was developed and applied to computer-based alarm systems, we note that it could easily be generalized and applied to any computer system with a significant role in a site's safeguards and security system.

REFERENCES

1. CG-SS-2: *Classification Guide for Safeguards and Security Information*, U.S. Department of Energy, Office of Classification and Technology Policy, Washington, DC (July 1990).
2. Lawrence Livermore National Laboratory, *Argus: A Sophisticated System for High-Security Facilities*, UCRL-TB-105443 (January 1993).
3. Simmons, Gustavus J., ed., *Contemporary Cryptology: The Science of Information Integrity*, New York, NY: IEEE Press (1992).
4. Chritton, Michael R., "Line Supervision of Alarm Communications," *Proceedings of the 32nd Annual Meeting of the Institute of Nuclear Materials Management*, Vol XX, pp. 557-62 (July 1991).
5. Sichertman, A., T. A. Renis, and R. A. Saleh, "Validating Detection Probabilities for the ASSESS Insider Module," *Proceedings of the 31st Annual Meeting of the Institute of Nuclear Materials Management*, Vol XIX, pp. 701-07 (July 1990).

**ASAM:
A Security Certification and Accreditation Support Tool
for
DoD Automated Information Systems**

Loreto Remorca, Jr. & William Barr Secure Solutions, Inc. 9404 Genesee Avenue, Suite 237 La Jolla, CA 92037	Robert Zomback U.S. Army CECOM, Space and Terrestrial Communications Directorate Ft. Monmouth, New Jersey 07703-5203 AMSEL-RD-C3-IS-P, #26	V. Michael Caputo MICON Services Company P.O. Box 1398 Eatontown, New Jersey 07724
---	---	---

Abstract

This paper provides an introduction to the All-purpose Security Assessment Model (ASAM) automated tool. ASAM is an on-going effort to define and implement an automated tool to support the security certification and accreditation efforts of the Department of Defense (DoD) automated information systems. ASAM uniquely integrates key aspects of the DoD certification and accreditation process that have been absent from most other security evaluation / assessment tools. In particular, ASAM addresses security issues across the entire system life-cycle, incorporates generic accreditation as defined by AR 380-19 and integrates a generic threat model.

This research and development effort was supported by U.S. Army Communications-Electronics Command, Space and Terrestrial Communications Directorate under Contract No. DAAB07-89-A050. This paper describes the initial ASAM efforts which are currently focused on the development of the Risk Management Review (RMR) portion of the ASAM.

Introduction

The U.S. Army Communications-Electronics Command (CECOM), Space and Terrestrial Communications Directorate, has identified the need to develop an automated tool to assist security analysts in the ongoing work of accrediting Army Tactical Command and Control System (ATCCS), telecommunications and automated information systems. This paper presents the requirements and rationale for the top level design of the All-purpose Security Assessment Model (ASAM) automated tool.

The earliest vision for the ASAM tool was to establish a process tailored to the risk assessment requirements of the tactical battlefield environment. Though ASAM still retains its original focus, it has evolved and expanded to support the more general objective to define and develop an automated framework for a comprehensive security certification and accreditation process.

At the heart of the ASAM tool design is the automation of the manual methodology currently in use to develop the generic accreditation documentation for the U.S. Army's Lightweight Tactical Fire Direction (LTACFIRE), Reserve Component Automation System (RCAS), and Fire Support Ada Conversion (FSAC) systems. In ASAM, this function is defined as the qualitative risk assessment that is part of the overall RMR process. The remaining ASAM functions serve to support the remaining activities in a security certification and accreditation process.

ASAM Overview

With the constant advances in computer and telecommunication technology, security is becoming an ever increasing area of concern. The DoD has addressed these concerns with the establishment of applicable evaluation methods, standards and regulations.

The DoD risk assessment process has evolved over many years. This evolution is most recently reflected in the Army's update to AR 380-380, AR 380-19. AR 380-19 introduces the concept of a generic accreditation approach which permits accreditation of systems to be fielded at many locations. The application of the generic accreditation process has the following beneficial effects:

- Lowers the overall system cost
- Decreases the time to field tactical systems
- Provides a higher level of security expertise during the accreditation process.

The ASAM approach offers benefits that are not realized when conducting a typical accreditation. These benefits include the ability to address security concerns throughout the development environment to overcome the following shortcomings:

- During the development process changes may be made to the initial design
- Developers may not track security requirements or be aware of them
- Implementation may not be verified against the DoD security requirements
- System changes necessitated after implementation are not cost effective.

These and other shortcomings indicate that the current security analysis process does not identify the security related problems at the appropriate times in the development life-cycle process. This failure results in the following negative impacts:

- A relatively high cost to correct the problems
- Delay or cancellation of the project
- An undesirable change in the system security mode of operation
- A decision by the Designated Accreditation Authority (DAA) that the security posture of the system is not acceptable and must be enhanced.

The recommended systems engineering approach to correct these failures is to involve the security analyst as a member of the development team from its inception and throughout the development process. Security related issues and recommendations that are addressed early on are most likely to result in an accredited system that meets the needs of the DoD without incurring any unnecessary costs.

ASAM Requirements

The requirements for ASAM have been derived from the early ASAM studies and analyses, open discussions during ASAM reviews, feedback from an early ASAM prototype (non-functional) and the "lessons learned" from ATCCS and RCAS accreditation efforts. One study of particular interest, conducted early in the ASAM project, was a review of the current state-of-the-art in automated risk assessment tools. The results from this study were influential in defining initial objectives for ASAM and was published under contract as the "*Risk Assessment Tool Evaluation*" report. The minimum ASAM general and system requirements are described in the paragraphs to follow.

General Requirements

During the ASAM analysis phase, the minimum certification and accreditation process requirements for an automated tool applicable to automated information systems were determined to be as follows:

- Provide environment-specific support
- Provide full life-cycle support
- Incorporate applicable mandates and requirements
- Address trusted system issues.

Environment-specific Support

Many automated information systems generate unique operational requirements because of its surrounding environment. In particular, military tactical systems possess garrison and deployed mode system operations and maintenance requirements that must be addressed for accreditation. Adoption of the a well defined Army RMR process will allow ASAM to readily address these unique factors.

Life-Cycle Support

The key to ASAM is the capability to perform evaluation and compliance activities throughout the system life-cycle that will support the certification and accreditation process in its entirety. The system development life-cycle defined in ASAM is depicted in Figure 1 below.

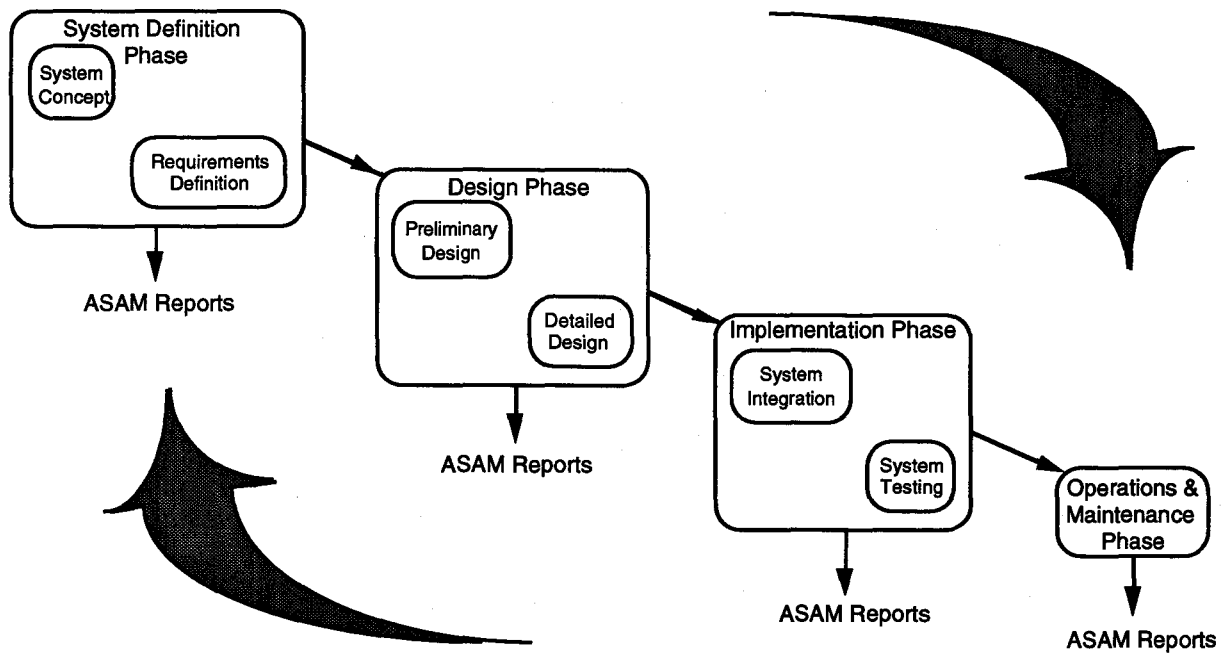


Figure 1. ASAM Application Across System Life-Cycle

In many cases, ASAM will be used to perform risk assessments of systems that are already operational. However, by using ASAM in the earliest stages of system development, system design deficiencies can be detected and corrected with the least impact. As ASAM is used throughout the system life-cycle, ASAM will effectively and efficiently gather and report the security assurance information required to complete the certification and accreditation processes.

Security Mandates and Requirements

Systems that require accreditation most often must meet mandated Government security requirements, greater program requirements, as well as specific mission security requirements. As such, pre-specified Government security requirements are incorporated into ASAM to provide compliance tracking for the security analyst. ASAM will also allow for the addition of program- or mission-specific and other security or functional requirements that are not able to be pre-specified in an ASAM database.

Trusted System Issues

The ASAM design philosophy treats confidentiality, integrity, and service assurance as security policy statements which require the implementation of protection mechanisms and assurances for their enforcement. ASAM will incorporate the DoD Trusted Computer System Evaluation Criteria (DoD 5200.28-STD) "Orange Book" requirements for each level of trust. In particular, ASAM will address the Orange Book requirements for trusted system accountability and assurance. ASAM will model the Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments (CSC-STD-003-85) "Yellow Book" concept of the interrelationship between the security mode of operation and the risk index.

In addition to the COMPUSEC issues above, ASAM will address and incorporate INFOSEC issues in the areas of:

- Procedural Security
- Personnel Security
- Physical Security.

System Requirements

Since the completion of the ASAM analysis phase, ASAM specification / development efforts have focused on developing a richer understanding of the Army generic accreditation process. The Army generic accreditation process has evolved and matured since the inception of the ASAM project. In response, the ASAM generic accreditation model has been adapted to meet these changes.

To achieve a balanced ASAM RMR process, the qualitative RMR was supplemented by the inclusion of a quantitative risk assessment. The qualitative-side of the assessment provides for consistent estimates of risk supported by descriptive statements of justification. The quantitative assessment provides for quick system-level security evaluations that can serve to refine the qualitative assessment focus.

In addition, requirements arising from the needs of the security analyst during the system life-cycle became apparent. These include the following:

- Timely decision making by project management
- Assurance documentation tracking
- Security requirements tracing.

Accreditation Model

Accreditation is the formal declaration by the DAA that a specified automated information system (AIS) is approved to process classified information at a specific security mode of operation using a prescribed set of safeguards. The process of performing the necessary assessments and preparing the documentation necessary for the DAA to make a decision is called the accreditation process. In a generic accreditation, the DAA approves all aspects of operation of an AIS except those that are specific to the location and operational environment of the system.

The ASAM generic accreditation model, used during the LTACFIRE, RCAS, and FSAC programs, is illustrated in Figure 2. Unique aspects of the ASAM generic accreditation model are its early analysis of the threat environment and its identification of user-defined system security requirements.

ASAM will guide the analyst in performing an evaluation of the system's technical and non-technical security features to establish the extent to which it meets the security requirements based upon the threat model described in the next section. ASAM will also generate security requirements analysis data that when combined with the threat analysis will feed into the certification.

ASAM will then conduct a RMR to assess the level of threat, system vulnerabilities, and estimated risk based on a correlation of the threats and vulnerabilities. The RMR includes recommendations for additional countermeasures needed to reduce risks that are considered to be unacceptably high.

Finally, a Users Security Manual/Standard Operating Procedure (USM/SOP) is developed to delineate the guidelines for the safe operation and location of the system. ASAM identifies those areas which may require a SOP to be generated. These SOPs are then expanded upon manually by the accreditation team.

It should be noted that the process just described is normally performed cyclically with feedback from later stages providing additional inputs into earlier stages as time moves forward.

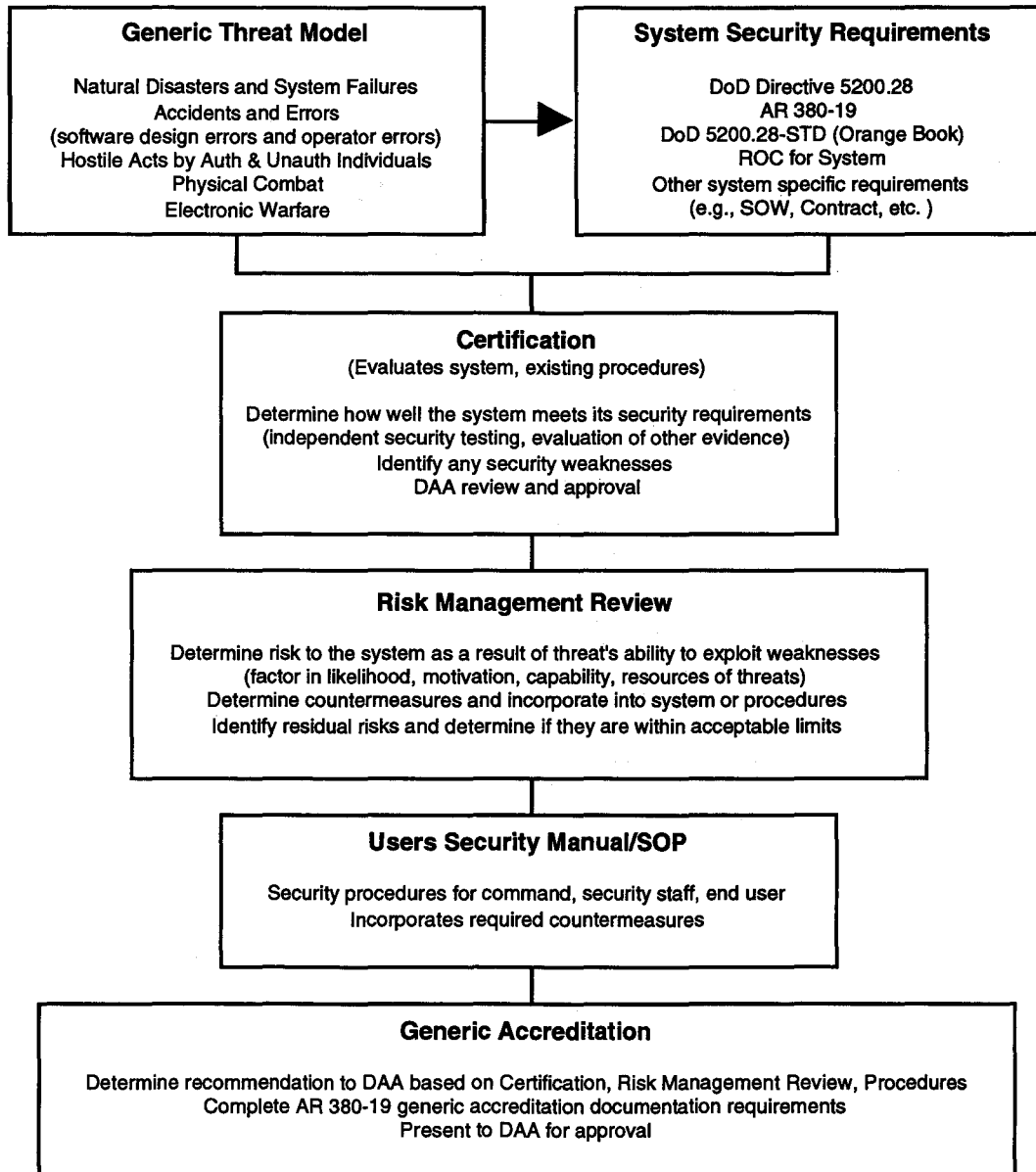


Figure 2. ASAM Generic Accreditation Model

Generic Threat Model

The ASAM RMR process is driven by the generic threat model depicted in Figure 3. Generic threats were identified based upon system threat assessment reports from a variety of tactical, telecommunications, and automated information systems. The threat model was developed to account for generic threats in both non-combat AISs and battlefield automated systems (BAS). Potentially, all threats in the figure could be brought to bear against a system.

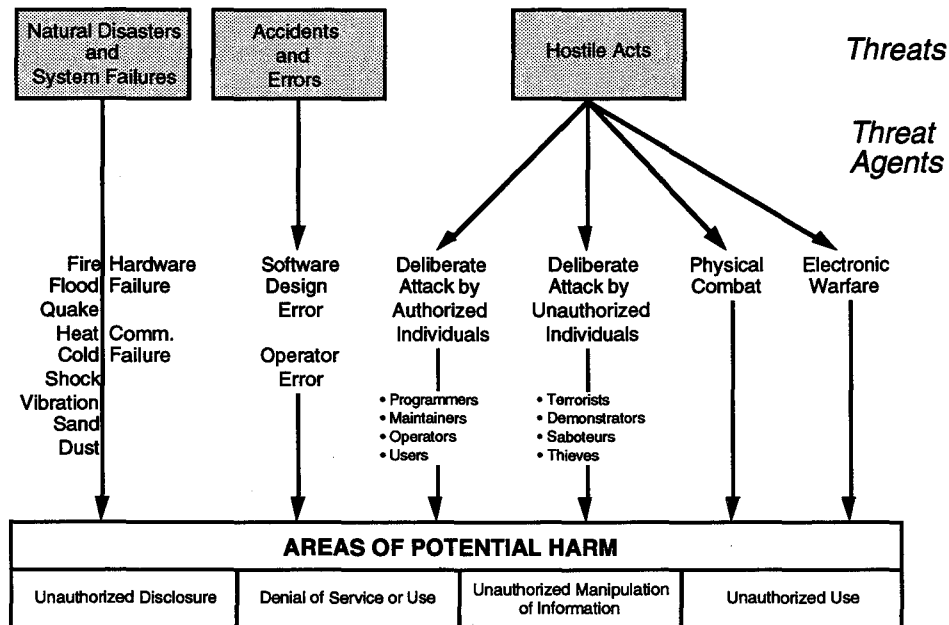


Figure 3. Generic Threat Model

Risk Assessment Process

AR-380-19 prescribes a risk management review to be conducted during the generic accreditation. The risk management review consists of the following four steps:

- Perform a risk assessment
- Identify and implement countermeasures
- Determine whether or not risks are acceptable
- Commit to on-going risk management.

The risk assessment is based on an analysis of threats to a system, identified system vulnerabilities to these threats, and the magnitude of the resultant risks. The risk assessment enables the security analyst to determine what countermeasures must be implemented to result in acceptable levels of risk for each area with residual risk. Finally, the requirements for on-going effectiveness reviews are determined.

The ASAM Risk Assessment Process is based on the generic accreditation process developed for Army certification and accreditation is illustrated in Figure 4. The risk assessment is driven by expected threats to the system, and by the embedded system security features. Therefore, the top layer of the risk assessment process depicts both the requirements that drive the system design, as well as the threats that can potentially be brought to bear against the user requirements-driven solution.

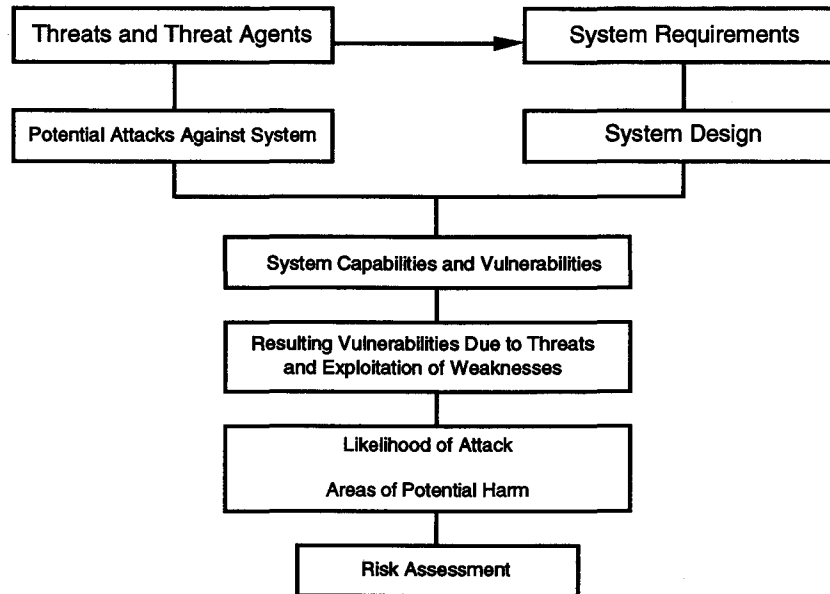


Figure 4. ASAM Risk Assessment Methodology

Generic Threat Model-Driven Risk Assessment

ASAM initially determines a subset of threats from the Generic Threat Model that can take advantage of the system vulnerabilities. These threats are applied to each system life-cycle to determine the level of risk. The ASAM risk assessment process subdivides the system life-cycle into the following phases:

- Development and production
- Fielding
- Garrison operations
- Exercises
- Combat operations
- Post deployment support and maintenance.

A quantitative risk assessment is available to the security analyst to perform this initial threat evaluation. This quantitative approach supplements the overall qualitative approach of ASAM by providing a high-level security assessment to validate the security requirements of the tactical system.

Threats include natural disasters and system failures, accidents and errors, and hostile attacks, such as deliberate attacks by authorized or unauthorized personnel and combat activities. These threats and threat agents, illustrated in Figure 2, are evaluated, and potential attacks against the system are assessed and documented.

Security Requirements Driven Risk Management Review

The ASAM risk assessment methodology also involves a security requirements-driven risk management phase. This approach applies threats against system security requirements. If the system is judged to meet a specific security requirement, the risk is considered to be minimal or none. If the system does not meet a specific security requirement, or it can not be determined that the requirement is met, then the risk is considered for further evaluation in the RMR process.

The system design is evaluated during the risk assessment. The accreditation team performs a security certification to ensure that all system security requirements are met, and if not, documents the identified design deficiencies for consideration in the risk assessment. The results of the system certification are then considered to judge whether or not the system is at risk due to the specific weaknesses in security features and functions determined during the certification. Threats and threat agents are resisted by the system, by the security practices and SOPs, and by the total operational environment within which the system operates.

Security Capabilities and Vulnerabilities

The threat-driven and security requirements-driven portions of the assessment methodology result in an enhanced understanding of the system capabilities and vulnerabilities. As a result of these analyses, the accreditation team develops a greater understanding of:

- System security capabilities and features
- System certification results – how well the system meets its security requirements
- The ability of threats and threat agents to exploit shortfalls identified in the system certification
- System operational procedures, security SOPs, and other countermeasures.

Based on this enhanced understanding, the accreditation team identifies the resulting system vulnerabilities due to threats and their potential exploitation of system security weaknesses. The system security shortfalls represent areas of potential vulnerability that may be exploited by the potential threats. Threats may attempt to exploit system security deficiencies during any life-cycle phase. If the system is vulnerable to a potential threat, further evaluation determines if the capabilities of the system or the controls and procedures built into the operational environment can mitigate the threat.

Estimating Likelihood of Attack and Residual Risk

Threats that survive this vulnerability assessment by being able to penetrate or bypass system security features or exploit system weaknesses are then evaluated to determine the likelihood that the threat would cause harm to the system. "Likelihood of attack" is a function of the motivation of the threat agent and its associated capabilities and resources to carry out the hostile act. Areas of potential harm include unauthorized disclosure, denial of service or use, unauthorized manipulation of information, and unauthorized use. The likelihood of harm to the system is derived from this analysis. The qualitative evaluation, or "sum" of the individual vulnerabilities determines the overall system

vulnerability. Finally, the likelihood of attack and the likely severity of the potential damage are combined to determine the residual risk.

Final Risk Assessment

Finally, the system residual risks are rank ordered by likelihood and potential damage. A determination is made as to whether or not the risk can be eliminated by identifying potential security controls and countermeasures. Technical/cost/schedule/benefit implications are considered to ensure that a sufficient return on investment can be realized through addressing the specific risk. This will lead to the correction or change to a specific system security feature or function, or the implementation of additional physical or procedural security controls. The residual risks are then evaluated to determine whether they can be eliminated or reduced to an acceptable level.

Threat agents can attack system assets (e.g., information, hardware, and software) at any time during the system life-cycle. The likelihood of a threat agent attack against a system will vary according to the current life-cycle phase. The effectiveness of countermeasures will also vary depending on the life-cycle phase. Thus, the level of risk is assessed separately for each of the system life-cycle phases.

ASAM Process Flow

The ASAM automated tool provides an application framework allowing for the incorporation of additional accreditation functions in the future. Figure 5 depicts the normal order of processing within ASAM. The shaded area highlights the initial ASAM development to provide the security analyst with the ASAM RMR (RMR) portion of the automated tool.

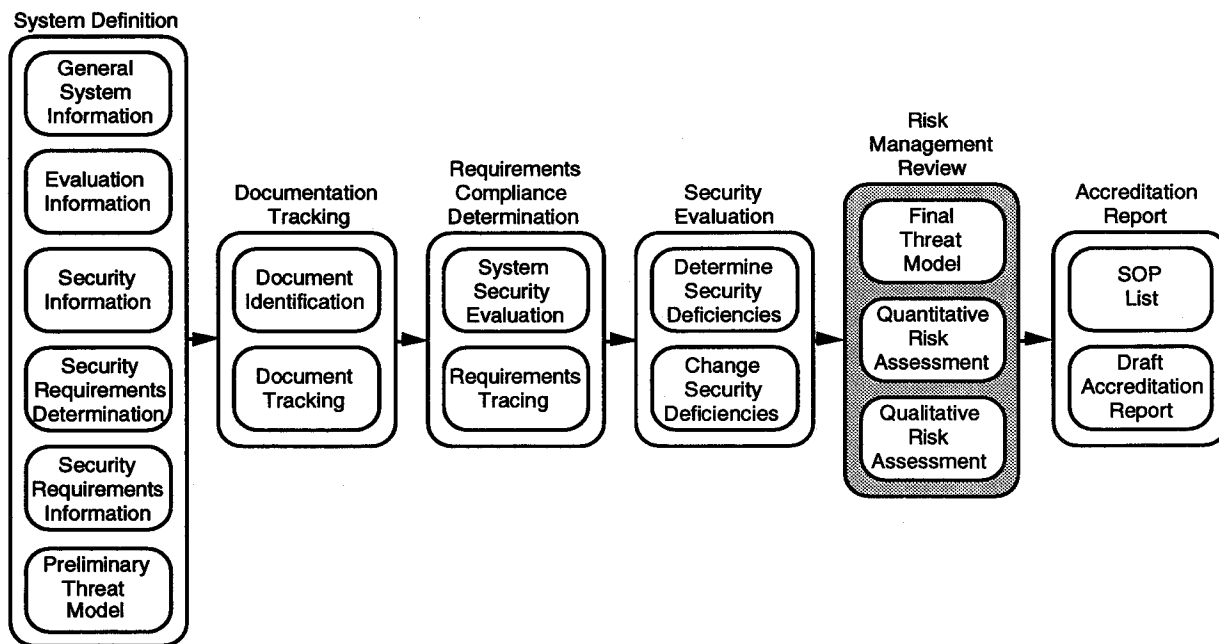


Figure 5. ASAM Process Flow

Since the security assessment process is iterative throughout the development life-cycle, ASAM is designed to provide the security analyst with a tool that is sensitive to these life-cycles and that is tailored to issues specific to each life-cycle phase. ASAM will provide the analyst with a minimum set of reports as follows:

- System Description
- Generic Threat Model
- Standard Security Requirements
- Documentation Requirements
- User-Defined Security Requirements
- Requirements Deficiencies
- Vulnerabilities List
- Final Threat Model
- Qualitative Risk Assessment
- Quantitative Risk Assessment
- SOP List
- Draft Accreditation.

System Definition Function

The System Definition function gathers preliminary system description data. ASAM uses this data to determine the initial system description, security requirements, and preliminary threat model. This system information is used to derive the "Yellow Book" TCSEC class and specific security requirements.

Documentation Tracking Function

The Document Tracking function determines the standard security documents required for the system being evaluated and provides a document tracking mechanism to follow the scheduled delivery / availability of these documents. Though missing documentation is an obvious deficiency, in many instances, delays in availability of security documentation can adversely affect the system development process if not addressed in a timely manner.

Requirements Compliance Determination Function

The Requirements Compliance Determination function tracks and examines the nature of compliance of individual security requirements that need to be satisfied / implemented. In the early stages of the life-cycle, the Requirements Compliance Determination function may simply examine compliance with a specified requirement. Analyzing the same requirement in later stages may require elaboration to provide documented assurances that the requirement is effectively implemented / tested.

Security Evaluation Function

The Security Evaluation function identifies an initial list of vulnerabilities based on the requirements deficiencies of the system being evaluated. This list can then be modified / updated as necessitated by the system and its system environment. At a minimum, ASAM should include the following list of vulnerabilities to begin the security evaluation:

- Access to trusted source code not properly controlled
- System not properly protected from theft
- Lack of automated security features
- Interconnection of systems processing information at different classifications
- Connection to unencrypted transmission channels.

Risk Management Review Function

The Risk Management Review function finalizes the threat model and performs both a quantitative and a qualitative risk assessment. The quantitative process is based on guidelines for computer security risk analysis that are compatible with international information technology security evaluation criteria (ITSEC) standards. The qualitative process is based on the manual methodology used to currently accredit the Army tactical and information systems.

ASAM uses the quantitative method to aid the analyst in focusing security evaluation efforts and to refine the preliminary threat model. The final threat model will then feed directly into the qualitative method. With this input, the qualitative process can immediately highlight those threats found to be significant during the quantitative process. As the development progresses, the periodic reviews of the quantitative and qualitative assessments will uncover and identify other security deficiencies.

Accreditation Report Function

The Accreditation Report function produces a list of the standard operating procedures (SOPs) that are required, as well as a draft accreditation report. The SOP list will be used to ensure that the final SOP meets the system requirements. At a minimum, ASAM should specify the following items for inclusion in the SOP:

- Access controls
- Handling of security incidents
- Requirements for a security training and awareness program
- Handling of classified data
- Emergency destruction of the system.

Acknowledgments

The authors of this paper wish to thank NSA and U.S. Army CECOM for its support and sponsorship of this effort. We also wish to acknowledge the following individuals for their contributions to ASAM: Mr. Michael Ware, Mr. John Preusse, Mr. Jeff Vignes, Mr. Kym Blair, Mr. Thomas Haley and Mr. Lester Lanphear.

A FINANCIAL MANAGEMENT APPROACH FOR SELECTING OPTIMAL, COST-EFFECTIVE SAFEGUARDS UPGRADES FOR COMPUTER- AND INFORMATION-SECURITY RISK MANAGEMENT¹

Suzanne T. Smith, Barranca Inc., POB 1349, Los Alamos NM 87544
Stephen Gale, University of Pennsylvania, 200 S 36th St, Philadelphia PA 19104
William J. Malampy, Univ. of Pennsylvania, 200 S 36th St, Philadelphia PA 19104

ABSTRACT

Just because yesterday's enemies have become today's friends does not imply lessened security needs. Moreover, today's tight money requires achieving security intelligently, no more than needed, and within budget. We explain how to achieve optimal risk management for computer- and information-security systems with a new software tool integrating two existing approaches: LAVA (the Los Alamos Vulnerability/Risk Assessment system, developed by Smith at Los Alamos National Laboratory) and VAM (the Value Added Model), developed by Gale et al. at the University of Pennsylvania's Wharton School). The LAVA/VAM amalgamation is a comprehensive tool for optimally upgrading security systems, based on potential exposure to loss (risk), subject to budget constraints. The combined tool calculates loss exposures as a function of the weaknesses or vulnerabilities in the existing baseline safeguards (countermeasures) system, the current threat strength, and the impact (cost) on the organization of successful threat attacks. Its reports describe vulnerability, threat, and loss exposure values in monetary and linguistic terms; these reports also give a linguistic cost/benefit basis for implementing strategies for upgrading missing or inadequate countermeasures. Then it identifies in monetary terms the baseline system's net present value (NPV) to the organization; a monetary cost/benefit analysis on possible upgrade strategies calculates the strategy's NPV and an internal rate of return (IRR) for the possible strategies. Optimal strategy candidates maximize NPV and have IRRs at least as large as the opportunity cost of capital, benefit-to-cost ratios greater than unity, and minimized vulnerability and loss exposure in areas most of interest to the organization.

Keywords: Risk management, vulnerability assessment, threat assessment, financial management, cost/benefit analysis, net present value, internal rate of return, optimal strategy, cost-effective safeguards upgrades.

¹Copyright © 1994 S. T. Smith, S. Gale, and W. J. Malampy. Permission is granted to the National Computer Security Conference Committee for publication and release of this paper in the proceedings of the 17th National Computer Security Conference. All other rights reserved.

I. Introduction

This paper explains how to achieve optimal risk management for computer- and information-security systems with a new software system¹⁻² that integrates two existing tools: LAVA³⁻⁷ (the Los Alamos Vulnerability/Risk Assessment system, developed by Smith at Los Alamos National Laboratory) and VAM⁸⁻¹² (the Value Added Model, developed by Gale and Malampy at the University of Pennsylvania's Wharton School).

There are four sections beyond this introduction. The first three sections are overviews of how LAVA's computer and information security model works, how the VAM segment works, and how the integrated LAVA/VAM Portfolio system works; the final section contains conclusions.

II. The LAVA Segment

LAVA, a systematic approach to vulnerability and risk assessment, examines in detail three separate entities: first, the weaknesses (or vulnerabilities) in the safeguards system protecting a group of assets, owned or valued by an organization, from a set of undesirable occurrences or outcomes that can be caused by a threat or set of threats exploiting the system weaknesses; second, the strength of the current threat against the system; and third, the potential costs to the organization if any or all of the outcome set should occur as a result of some threat attack.

LAVA has three distinct parts. The first part is the mathematical model upon which the rest of the LAVA system is built; this model is based on classical risk assessment theory, utility theory, expert systems theory, cognitive science, hierarchical systems theory, fuzzy set theory, possibility theory, and natural language processes. The second part is a general software system, an expert system framework, implementing the mathematical model as a user-friendly software engine (or computer program) that drives a variety of application systems. The third part is made up of the several application system models, each of which is a knowledge-based expert system, whose data appears to the user as sets of interactive questionnaires and sets of instruction screens. The most familiar of these LAVA application systems is LAVA/CIS, the application for Computer and Information Security that has been in use throughout the federal government since 1984; a newer, related application system is LAVA/LAN, the application for Local Area Network security.

Each LAVA application system has, as basic components, sets of *assets*, *threats*, and *undesirable outcomes*. LAVA defines an asset as something of value to the organization that must be protected from harm or compromise. There are essentially two kinds of assets: tangible assets and informational assets. For LAVA/CIS and LAVA/LAN, there are four asset sets. The first is the *facility*, defined as personnel and the physical plant—whatever constitutes the base of operations. The second is the computer *hardware*, including computers and ancillary equipment like printers, plotters, data communication devices, terminals, and so forth. Third is all *machine-interpretable information*, which is coded or encrypted information that requires machine translation before it makes sense to a human, such as software, cad-cam files, computer or word-processor data files, encrypted messages, and so forth. Fourth is all *human-interpretable information*, encompassing documents (such as phone book, personnel lists, schedules, engineering drawings, itineraries and routings, maps, organization charts, other charts, plots, diagrams, printed computer output, letters, reports, scratch notes, and so forth), video and computer-screen displays, other displays (such as blackboards, strategy boards, bulletin boards, status boards, and the like), and conversations.

LAVA defines threats as active forces (human forces or forces of nature) that can harm or compromise the organization's assets. Most LAVA application models have three threat categories, each having several subcategories. We first consider all *natural and random hazards*,

including unintentional human error (handling, maintenance, poor training) or equipment malfunction; widespread (or regional) major hazards, including catastrophic fire, flood, seismic activity, volcanic disasters, severe storms like hurricanes or tornadoes, and so forth; ordinary fire damage; ordinary water damage; power abnormality damage; heating, ventilating, or cooling damage; maintenance or housekeeping damage; and so on. Next, we consider the *onsite human adversary*, assuming malice aforethought on the part of the adversary, who is able through legitimate or illegitimate means to pass through or evade normal physical security at the organization's physical outer perimeter and is on site; this threat includes disgruntled employees; authorized insiders; hostile intelligence services; industrial spies; terrorists; paramilitary organizations; mentally unstable persons; political activists; the press; and so forth. Finally, we consider the *offsite human adversary*, again with malice aforethought assumed, but whose location is exterior to the organization's physical outer perimeter; similar to the previous threat except the perpetrator is located outside the physical outer perimeter of the organization's property, it includes hostile intelligence services; industrial spies; terrorists; paramilitary organizations; mentally unstable persons; political activists; the press; and so forth.

Each human threat agent may have one or more goals as part of the attack agenda. Some of these goals may include sabotage, unauthorized control of information or other assets, ransom or extortion, political or economic advantage, political unrest, adverse publicity or loss of reputation for the organization, publicity for a cause or themselves, breakdown in international relations, strategic advantage, revenge, surmounting an intellectual challenge, fomenting revolution or war, or satisfying an irrational need resulting from general insanity

Undesirable outcomes are the unwanted results of a successful attack by a threat or combination of threats against the organization, the facility, or one or more of the organization's assets. For both LAVA/CIS and LAVA/LAN, there are six undesirable outcomes: *unauthorized access or use*; *theft* (or abduction for personnel asset); *damage, tampering, or unauthorized modification* (or physical injury or mental tampering, for personnel asset); *destruction* (or death, in the case of personnel asset); *unauthorized disclosure*; and *unavailability or denial of use* (which can include hostage or siege situations causing unavailability). LAVA models the degree of possibility of each outcome's occurrence for each threat-asset pair, and keeps track of these values in a fuzzy outcome possibility matrix. It is important to note that a single event can result in the simultaneous occurrence of more than one of the outcomes.

Impacts or costs to the organization of attacks, whether successful or not, are measured in both monetary and nonmonetary (or linguistic) terms, and come about from a variety of cost factors. Monetary cost factors are calculated for such things as investigation and followup, replacement, repair, retraining, business interruption, loss of future business or funding, business or contract termination, litigation, fraud or embezzlement, and so forth. Nonmonetary cost factors can arise from such potential events as disruption of international relations, organizational embarrassment, loss of strategic posture, loss of public trust or confidence, loss of reputation, organizational stress or disruption, loss of morale, invalidation of treaties, loss of future funding, unsafe or restrictive working conditions, and so on.

Each outcome can occur in varying degrees of severity for each [threat, asset, safeguards function] combination. Outcome severity depends on the degree of possibility of the outcome for each [threat, asset, outcome] combination as expressed in the outcome possibility matrix, the organization's value structure with respect to the outcome set (an organization may have a greater aversion to one outcome over another), the strength of the current threat, the value of the asset to the organization, and the relative weakness of the safeguards function (the performance measure of the safeguards function, which is called "vulnerability measure" in the LAVA reports).

Having established the threat, asset, and outcome sets and the outcome possibility matrix, we then address what constitutes the "ideal" safeguards system (the one you would have if neither time nor money were considerations) for preventing the threats from attacking the assets

and achieving the postulated outcomes.

LAVA's safeguards system model represents the kinds of controls needed to protect the assets from the threats. A set of control functions, called Safeguards Functions, is defined for each discrete pair of threat-asset combinations in such a way as to provide complete protection if the combined degrees of functional control are adequate for the sensitivity of the operation. For this we define a set of safeguards functions for each of the distinguishable threat-asset pairs so that the relative importance of each function within the set of functions for each T-A pair is about the same. The degree of functional control is determined by aggregating the levels (or degrees) of completeness of the control subfunctions, or Safeguards Subfunctions. For *each* of the individual safeguards functions, we define a set of subfunctions that provide performance criteria for the adequacy and completeness of that safeguards function; each of the subfunctions is devised so that the relative importance of a subfunction within a specific function is about the same. Completeness of each safeguards subfunction is determined by aggregating the completeness scores of the Safeguards Elements (specific safeguards, protective measures, and countermeasures associated with each safeguards subfunction) and their Attributes (characteristics of the individual countermeasures that contribute to their strength and completeness).

The safeguards elements and attributes are modeled as questions in one of LAVA's interactive questionnaires (the Vulnerability Assessment Questionnaire); each element and attribute is related to the safeguards system model as an element of control necessary for the completeness of one or more, often many, of the safeguards subfunctions. LAVA's analysis of the data thus gathered establishes two things: what constitutes the baseline safeguards system used as input to VAM, and what vulnerability levels are inherent in that system.

Using several detailed questionnaires, LAVA collects the site-specific information from teams of persons associated with each site. The questionnaires and other application-specific data required for the software engine to operate upon arise from the existing safeguards orders; from inspection, evaluation, and certification criteria; from checklists and standard operating procedures; and from discussions with recognized experts in the field. LAVA produces automated reports detailing: what constitutes the existing safeguards system at the site under analysis; what safeguards mechanisms are candidates for use in upgrading the safeguards system; the strength of the existing threat; the vulnerability measures for each safeguards subfunction with respect to the existing threat; and what the potential loss exposure is for each [threat, asset, safeguards function, outcome] combination, in both monetary and nonmonetary terms.

Using LAVA's approach for vulnerability/risk analysis has benefits that do not accrue from the use of other, more quantitative, probabilistic methods. First, LAVA's automated report generators produce results that are immediately usable, both by managers who must make major, far-reaching decisions and by security personnel in the field whose job it is to maintain an acceptable level of safeguards. Second, because LAVA produces *both* qualitative and quantitative results, users feel more comfortable with the results because they understand both the results and the information that produced those results. Third, because LAVA does not require the user to generate probabilities (often unfounded) for its operation but instead relies on a natural-language, user-friendly interface to acquire its data, users are more willing to act upon its results. And finally, because of the team environment in which an assessment is performed and the discussions that arise among team members, the *process itself* of using a LAVA application has proved to be an experience that both raises the security consciousness of the users and enhances the overall working environment at the facility.

III. The VAM Segment

Originally, the purpose of the Value Added Model (VAM) was to answer questions about

the monetary value that security measures added to corporations. It determined which security strategies produced the highest financial return to an organization given past patterns of breaches and/or threats to corporate property (tangible or intangible). As such, VAM characterized the safeguards system risk problem as finding those security strategy(ies) that maximize the value added to the organization given specified risk level, and determining the value added (the benefits that the security system adds to the organization) in terms of the value of the losses *avoided* by a given security strategy relative to an existing or security strategy.

In the VAM software, we structure the analysis as follows. First, the security analyst defines and identifies the kinds and extent of the safeguards system risk problems that are of interest (or critical) to the organization. Then VAM selects one or more technically and operationally feasible security strategies that, from the perspective of the security analyst, are likely to prevent the attacks that cause risk or mitigate the potential losses once the breach has occurred. For each technically and operationally feasible safeguards system strategy, VAM next computes the expected losses, benefits and costs. Then VAM estimates the value of the security strategy by substituting these benefits and costs into its financial analysis (investment) component. After the VAM software has analyzed all of the combinations of technologies and operational procedures, it makes a ranking according to each of the three investment criteria—NPV, IRR, and Benefit/Cost Ratio. Finally, VAM generates a report describing the ranking of technologies and operational procedures, and prepares and presents to management their financial projections for final decision-making on investment.

VAM has five operational modules (Problem Definition, Exposure and Breach Analysis, Security Component Feasibility Analysis, Formulation of Security Strategies, and Estimation of Breach Occurrence) and four financial modules (Expected Loss Estimation, Expected Benefit Estimation, Cost Estimation, and Investment Analysis). VAM ranks candidate security strategies according to their relative value added to the organization using three measures of investment analysis—Net Present Value (NPV), Internal Rate of Return (IRR), and Benefit/Cost (B/C) ratio.

NPV quantifies the value added of safeguards system strategies by computing the sum of the present value of each strategy's stream of expected benefits (net of annual operation and maintenance costs generated over the economic life of the strategy) less the initial purchase and installation costs. Using this measure, the investment decision is simple: invest in the S&UCS strategy that has the largest NPV (given that the NPV is greater than zero).

IRR is the discount rate that sets the NPV of a safeguards system strategy equal to zero. Specifically, IRR is the discount rate that equates the present value of the security strategy's net benefits (including interest earned from investing these benefits) and the initial purchase and installation costs. The investment decision rule for accepting a strategy is again simple: accept any strategy whose IRR is at least as large as the organization's opportunity cost of capital, as set by the chief financial officer or other accepted authority.

B/C ratio measures the safeguards system strategy's "bang per buck invested". B/C ratio is the appropriate measure of the value(C)added of a security strategy when the size of the capital budget is the principal constraint in the resource allocation process. The investment decision rule for accepting a security strategy is to accept any strategy whose ratio is greater than unity (1).

IV. How Does the LAVA-VAM Portfolio Model Work?

The LAVA-VAM joint approach, called the Portfolio Model, incorporates LAVA's critical outcome and vulnerability estimation procedures with the VAM's methods of assessing investment decisions to provide an integrated model for the analysis of safeguards systems. LAVA elicits the parameters of the baseline security system and provides a relative vulnerability analysis (score) for all safeguards functions and subfunctions, its assessment of dynamic threats, and its

analysis of consequences through the use of the expert system model. VAM provides a means for assessing the financial value added of alternative countermeasure strategies relative to baseline conditions.

Three additional components of a comprehensive safeguards system model make the approach truly functional. First is a procedure/database that generates a comprehensive list of potential penetration scenarios for analysis. Second is a database of the available technical, operational, training, managerial and related support functions that constitute alternative countermeasure strategies *and* a procedure for selecting and combining them with respect to specific analytical criteria. Third is a user-friendly human/computer interaction process, one based on Microsoft Windows, that permits safeguards system analysts to use this system in a simple, straightforward manner under a variety of operational conditions.

This integrated model provides a system that calculates the optimal configuration of effective and efficient safeguards countermeasures for computer- and information-security systems. LAVA, although expressing loss exposure in both monetary and nonmonetary terms, operates without using a specific organizational budget constraint and so suggests, as candidate safeguards improvements, *all* safeguards elements it finds to be either missing or inadequate in performance. The VAM, on the other hand, currently uses financial indicators of expected loss functions as the basis for its calculations of investment decisions. In the Portfolio Model, LAVA's measure of vulnerability is used as the calculation base, and the optimal decision for investments in safeguards countermeasures is based on *maximizing the decrease in the assessed vulnerability of the system, while maximizing the value-added, given a specified budget constraint.*

As currently designed, the Portfolio Model for safeguards systems for computer- and information-security systems is based on the use of Microsoft Windows as the operating environment. The Windows screens facilitate navigation through the Portfolio Model and consist of commands relating to LAVA, VAM, the data collection and analysis procedures, and countermeasures database. The opening segment of the assessment includes material consisting of definitions, instructions, the assessment team, and basic data on the history, operation and spatial configuration of the site and the organization. The next step consists of a series of screens that facilitate LAVA's Vulnerability Analysis (VA) section, including the VA Interactive Questionnaire to be answered by the assessment team; the VA scoring; the preparation of the VA report, which shows the degree of weakness of each of the safeguards subfunctions and lists the missing and inadequate elements in the safeguards system; and the production of a file for analysis by the VAM containing all of the elements in the existing safeguards system.

Then the user sees another series of screens in which the VAM calculates the value-added of the existing safeguards system, including a relative valuation of the current safeguards system (that is, in terms of a comparison with other system elements, given the cost of each element of the safeguard system and its operation) and a comparison of the purchase and operating costs as determined from the database on available countermeasure strategies.

Next comes a series of MS Windows screens on LAVA's Dynamic Threat section, which determines threat strength with respect to [threat, asset, safeguard functions] combinations, generates a report listing the current threat strength showing the respective degree of vulnerability for each of the safeguards subfunctions, and produces a file of all the candidate safeguards elements (independent of cost and time) for VAM analysis.

Then a series of Windows screens guide the user through the Consequence Analysis section of LAVA (questionnaires, scoring, and automated reports), including the organization's evaluation of its assets (asset worth); the impacts of each possible successful attack for each [threat, asset, safeguards function, outcome] quadruplet (in both monetary and nonmonetary terms); and the loss exposure for each quadruplet (in both monetary and nonmonetary terms).

The next step of the Portfolio Model draws on both VAM and LAVA and the available countermeasures database. Initially, VAM calculates the value-added of the safeguards elements

that LAVA produces as candidates (that is, those that bring the vulnerability score as close to zero as possible). Next, it selects from the available countermeasures database additional candidate strategies for analysis subject to the budget constraint. Using each candidate strategy in successive iterations, LAVA calculates new vulnerability and risk measures for that strategy upgrading the safeguards system (that is, decreasing the vulnerability score). Finally, VAM calculates the value-added of each alternative safeguards combination to determine the optimal strategy--the one that produces the highest value-added and maximum decrease in vulnerability and risk, given the organization's specified budget.

The Portfolio Model's final product is a report identifying the elements of the optimal strategy for upgrading the safeguards system. The Portfolio Model generates the report by means of reference to a series of screens in Windows that allow for combining text, tabular, and graphical information. Under the current designs, the report is capable of providing detailed information on the scenario(s) analyzed and the variation in safeguards necessary to protect the site under varying conditions. Where appropriate, the report will also review additional financial assumptions including alternative cost conditions and budget constraints.

IV. CONCLUSIONS

The Portfolio Model is an integrated system made up of the key processes of LAVA and VAM; it also includes a database on available countermeasures and their costs. The entire system operates under a MS Windows-based control structure. The purpose of the Portfolio Model is to assist safeguards policy makers, designers, and engineers in selecting the most effective and cost-effective systems, given the need to simultaneously achieve efficiencies in investment patterns. The model design, moreover, both reflects the complexity of selecting modern safeguards systems and takes advantage of the existing state of the art with respect to the available analytical and decision support procedures.

What we believe we have done is to advance the state of the art from an adequate but primitive state where there are two approaches to the common problem (of intelligent computer- and information-security system upgrades to minimize vulnerability and risk, according to system sensitivity, within the organization's budget) to an advanced state with single model, the Portfolio Model, which produces the grounds for integrating considerations of effectiveness and efficiency in the selection of safeguards system upgrades for computer- and information-security systems.

REFERENCES

1. S. T. Smith, S. Gale, and W. J. Malampy, "LAVA/VAM: An Integrated System for Optimal Risk Management in Ultra-High Risk Facilities," Proc. 1993 Symposium on High-Consequence Surety: Academic Views, Albuquerque, NM (October 1993), BI-UR-93-002.
2. S. T. Smith, S. Gale, and W. J. Malampy, "Optimal Risk Management in Ultra-High Risk Facilities: LAVA/VAM—A New Approach," (BI-UR-94-002) Proc. 1994 International Conference on Fuzzy Logic, San Diego CA, September 1994).
3. S. T. Smith and J. J. Lim, "An Automated Method for Assessing the Effectiveness of Computer Security Safeguards," Proc. IFIPS Second International Congress on Computer Security, Toronto, Canada (September, 1984).
4. S. T. Smith and J. J. Lim, "Risk Analysis in Computer Systems--An Automated Procedure," *Information Age*, Vol. 7, No. 1, pp. 15-18 (January 1985).

5. S. T. Smith et al., "LAVA: A Conceptual Framework for Automated Risk Analysis," in L. A. Cox, Jr. and P. F. Ricci, eds., *New Risks: Issues and Management*, pp. 315-330. New York: Plenum Press, 1990.
6. S. T. Smith and M.-L. Jalbert, "LAVA/CIS Version 2.0: A Software System for Vulnerability and Risk Assessment," Proc. 13th National Computer Security Conference, Washington, DC (October 1990).
7. S. T. Smith, "LAVA/CIS Version 3.0a: A User's Guide (Draft)," Barranca Inc. Publication BI-UR-93-001 (June 1993).
8. S. Gale, K. Duncan, et al., "Conceptualizing a Value Added Approach to Security Management," *Security Journal*, 3 (1992), pp. 4-13.
9. S. Gale, K. Duncan, et al., "The Atkinson Value-Added Model," *Security Journal*, 3 (1992), pp. 14-26.
10. S. Gale, K. Duncan, et al., "An Implementation of the Atkinson Model," *Security Journal* 3 (1992), pp. 27-44.
11. S. Gale and W. J. Malampy, "Research on the Value-Added Model for Security Analysis," Proc. 39th Annual seminar of the American Society of Industrial Security, Washington, DC (August 1993).
12. S. Gale and W. J. Malampy, "The Relationship of the Value-Added Model (VAM) to LAVA and the Development of a Comprehensive Portfolio Model Incorporating LAVA and VAM in a User-Friendly Human/Computer Interaction Format," presented to the Symposium on High Consequence Surety (Academic Views), Albuquerque, NM (October 1993).

**THE ELECTRONIC INTRUSION THREAT
TO NATIONAL SECURITY & EMERGENCY PREPAREDNESS
TELECOMMUNICATIONS:
AN AWARENESS DOCUMENT**

AUTHORED BY: Dr. Joseph Frizzell
Office of the Manager, National Communications System
701 South Courthouse Rd
Arlington, VA 22204
(703) 692-0524
frizzelj@cc.ims.disa.mil

Mr. Ted Phillips (POC)
Booz-Allen & Hamilton Inc.
8283 Greensboro Drive
McLean, VA 22102
(703) 902-5420
(703) 902-3172 fax
phillips_ted@bah.com

Mr. Traigh Groover
Booz-Allen & Hamilton Inc.
8283 Greensboro Drive
McLean, VA 22102
(703) 902-5376
groover_traigh@bah.com

1.0 INTRODUCTION

This paper presents an assessment of the threat posed by individuals or groups that gain unauthorized access to elements of the nation's public telecommunications infrastructure. These individuals and groups, who will be referred to here as *electronic intruders*, currently possess the skills and tools to cause disruptions or denials of service, remotely monitor telecommunications traffic, steal sensitive information from network elements, modify network data bases, and commit fraud.

The United States Government is concerned with the threats from electronic intrusion because of its heavy reliance on public telecommunications networks to maintain communications in times of national emergency or crisis. Over 90 percent of the U.S. Government's telecommunications services are provided by commercial carriers. National security and emergency preparedness (NS/EP) organizations rely heavily on the public switched network (PSN)¹ to provide telecommunications services in times of national crisis or disaster.

This paper is based solely on open source information -- case histories, electronic intruder files, technical journals, and other freely-available data. The reasons for this approach are three-fold. First, the nature of this information creates none of the restrictions imposed by the use of classified or proprietary data. Second, intruders are quite prolific when writing about themselves and have generated hundreds of megabytes of data about their activities, all of which are available electronically. Third, the high level of interest by those outside the computer underground has resulted in a large volume of periodical literature and academic work focused on electronic intrusion techniques and incidents.²

2.0 TARGETS AND TECHNIQUES

Electronic intruders constitute a significant threat to the telecommunications infrastructure in the United States. They have made intrusions into key network elements, created denial-of-service problems, monitored communications, searched network data bases, and modified and deleted data base information. On at least two occasions, they have compromised NS/EP telecommunications services. Because of electronic intruders' increasing knowledge and skills in telecommunication systems, the threat to the PSN from electronic intruders is continually growing.

2.1 Degree of PSN Penetration

Electronic intruders have penetrated every category of network elements of the PSN including switching systems; transmission systems; operations, administration, maintenance, and provisioning (OAM&P) systems; signaling systems; and packet networks. Though obtaining accurate information

¹Because no single term is descriptive enough to encompass all the components and networks which comprise the U.S. telecommunications infrastructure, this paper defines the term *PSN* in a very broad manner. In addition to the switched network, this paper uses the term *PSN* to include virtually all terrestrial telecommunications systems and networks such as public data networks, cellular systems, and signaling networks.

² This paper's source material is listed in Appendix A. Readers are encouraged to request a copy of the identically-titled report upon which this paper is based from the Office of the Manager, National Communications System, Office of Plans and Programs, 701 South Courthouse Road, Arlington, Virginia 22205.

concerning the frequency of attacks on particular network elements is virtually impossible due, in part, to the vast majority of attacks not being reported, it is apparent that electronic intruders have regularly attacked all types of networks that are linked to the PSN.

Points of Intrusion. Carriers' internal *corporate networks*³ have been a fertile ground for exploitation by electronic intruders. One of the characteristics of the current telecommunications infrastructure is that all network elements are highly interconnected via carriers' internal corporate networks. This connectivity provides remote access to network elements for network engineers, technicians, craftsmen, vendors, and other legitimate users. Remote access to network elements is a double-edged sword -- it enables carriers to reduce operating costs, but it also provides many intrusion opportunities for electronic intruders.

Since important systems reside on carriers' corporate networks, significant security provisions are normally implemented. However, these countermeasures are usually employed around the perimeter of the network, at dial-in ports and gateways. Once legitimate users or electronic intruders are authenticated by these perimeter security points, they can attempt to connect to a wide variety of network elements and other resources.

The Poulsen Case. One recent case illustrates the potential for abuse posed by electronic intruders. On April 11, 1991, law enforcement authorities arrested Kevin Lee Poulsen in Van Nuys, California, 17 months after he was indicted on a variety of computer fraud and wiretapping charges. Poulsen, known by the handle *Dark Dante*, allegedly masterminded a complete computer and telephone system invasion. If the allegations against Poulsen are true, he was responsible for the most comprehensive, coordinated attack on the PSN ever documented.

Some of the allegations against Poulsen and his two accomplices are informative. Poulsen allegedly intruded on local exchange carrier (LEC) service provisioning systems numerous times. He allegedly modified existing telephone services, added new telephone services (some without billing), forwarded calls to other numbers, and dual-provisioned telephone lines. He allegedly intruded on LEC maintenance/test systems to electronically monitor telephone conversations. He allegedly intruded on LEC data bases and obtained telephone numbers (some unlisted), street addresses, customer names, and other sensitive data. He allegedly physically broke into carrier offices and stole equipment, software, identification badges, and other material. He allegedly compromised ongoing law enforcement investigations. He allegedly sold sensitive data obtained from LEC data bases. He allegedly illegally established or modified telephone services for other individuals. He allegedly manufactured false identification, including telephone company badges and drivers licenses. He allegedly intruded on other computer systems for profit, including California DMV, several credit bureaus, and an Air Force computer network.

The MOD Case. On July 8, 1992, several members of an electronic intruder group known as the Masters of Disaster (a.k.a. Masters of Deception, a.k.a. Masters of Destruction) were indicted on 11 counts including conspiracy, wire fraud, computer fraud, and interception of electronic communications. Some of the allegations against the members of MOD are informative.

³ Carriers' corporate networks are used to interconnect network elements in their traffic networks, to provide connectivity for remote network administration, maintenance, repair, and other functions.

The group developed and unleashed *programmed attacks* on telephone company computers. They monitored data transmissions on X.25 networks, looking for passwords and access codes. They illegally accessed PSN computers, created new circuits, added services with no billing records. They changed an adversary's long distance carrier to more easily obtain the adversary's calling records. They sold passwords and access codes for money and destroyed data in several computer systems. The MOD members arrested reached a plea bargain agreement and received sentences of over one year.

2.2 Information Gathering Activities

Electronic intruders have demonstrated a high level of competence in identifying, and gaining information on PSN network elements and services. *War dialing* is usually the primary method of information gathering for novice and developing electronic intruders. War dialing is accomplished by using a software tool that rapidly dials all numbers within a specified range. Electronic intruders frequently scan the 200, 55X, 57X, 95X, 97X, and 99X exchanges in given NPAs because this is where carriers frequently place their internal test and maintenance services. Once a dial-up access number is found to a system, the next step many junior electronic intruders take is *trashing*, or *dumpster diving*. Trashing is the activity of sorting through the trash dumpsters of telecommunication companies in order to find information such as system manuals, computer printouts, reports, discarded login account and password combinations, memoranda, telephone directories. Perhaps the most common information gathering tactic used by the more experienced electronic intruders is that of *social engineering*. Social engineering is the act of impersonating telecommunication employees or network security managers in order to gain valuable information including passwords, logins, and access numbers. The most dangerous, but least wide-spread threat in this area is *unlawful entry* into telephone company facilities. The Poulsen case illustrated the potential for exploitation posed by electronic intruders breaking and entering into LEC end offices.

2.3 Increasing Technical Sophistication

Electronic intruders are continually developing more creative and sophisticated methods for compromising various elements of the PSN.

Cellular/Wireless Attacks. As the use of wireless telecommunication services has exploded during the past decade, electronic intruders have sought to exploit these technologies. The primary concern pertaining to this type of attack is the disclosure of sensitive information. Cellular phones can be exploited by any person using a properly equipped police scanner (capable of monitoring the cellular frequency band, 824-894 Mhz). With this equipment, electronic intruders can easily monitor cellular transmissions and capture potentially sensitive data. This is especially important when cellular users pass credit card numbers, login/password data, access codes, or other sensitive data.

Packet Network Attacks. Data networks are an integral part of the nation's telecommunications system. Public packet switched data services offered by telecommunications carriers are one manifestation of these networks. Most carriers also use packet networks for their internal *corporate networks*. As discussed earlier, these networks connect operations, administration, maintenance, and provisioning (OAM&P) systems, switching systems, and other network elements together. Electronic intruders consider these PDNs and corporate networks to be fertile ground for exploration. Consequently, computer underground bulletin boards and publications contain an abundant supply of network user identifications (NUIs) and network user addresses (NUAs) for PSN

elements. One of the techniques invented by electronic intruders to avoid toll charges is called *weaving*. This involves dialing into a local computer or network, exiting the computer/network on an outdial line, connecting to another computer/network, and outdialing again to the target system. Weaving in and out of computers not only avoids toll charges, but it also makes tracing intruders difficult, if not impossible. Today, this technique is used by a vast majority of electronic intruders as a means to avoid arrest. Electronic intruders are also beginning to exhibit skills related to direct manipulation of data network devices (such as packet assembler/disassemblers, or PADs). These skills are noteworthy because of the reliance on packet switched networks for connecting network elements.

Advanced Software Techniques. In recent years, electronic intruders have begun to utilize advanced software techniques in their activities. This phenomenon is a natural outgrowth of basic electronic intruder activities, in which software tools play a large role. However, a different genre of software tools is being increasingly used by advanced electronic intruders. These tools are often custom-developed by members of the computer underground, and they are frequently distributed to other electronic intruders with both source and object code, allowing for quick and easy modification to suit specific tasks.

The most dangerous type of this software is new or modified code which the electronic intruders plant surreptitiously inside network elements. These small programs can be written to function like software viruses, worms, or trojan horses. Currently, there have been only a few documented cases of surreptitious code being planted in PSN network elements. However, if an attack of this type were successful, it could prove devastating to the PSN.

A second advanced software technique gaining popularity in the electronic intruder community is equally dangerous. This technique involves modifying legitimate software tools stolen from telecommunications carriers and equipment manufacturers. At least four well publicized incidents illustrate this problem: Kevin Mitnick, a.k.a. *Condor*, Herbert Zinn, a.k.a. *Shadow Hawk*, the Legion of Doom (LOD), and Leonard Rose, a.k.a. *Terminus*. In each of these cases, source code for network elements was stolen electronically and modified for purposes unintended by the owner. In these four cases, no PSN network element was compromised by the planting of modified source code. However, the skills demonstrated by these electronic intruders could easily be applied to the PSN.

Programmed Attacks. A highly sophisticated form of software attack has been detected several times in various networks, and is considered to be the leading edge of electronic intrusion activities. These are called *programmed attacks* because they rely on highly customized software programs that target specific types of computers or network elements. Little data has been gathered on these attacks because they are seldom detected. One apparent purpose for programmed attacks is for reconnaissance to gather information about the network. These programs can be assumed to be the result of significant prior effort on the electronic intruders part. In addition to the MOD case described above, the LOD case illustrates electronic intruder skills in this area. In 1990, several members of the Atlanta branch of the Legion of Doom were arrested on charges of penetrating and disrupting telecommunications network elements. Federal agents accused the LOD members of planting a series of destructive "time bomb" programs in network elements in Denver, Atlanta, and New Jersey. These time bombs were designed to shut down major switching hubs, but were defused by telephone company employees before causing any damage. The capability illustrated by this category of attacks has not fully matured. However, a coordinated attack using these types of tools directed at the PSN with a goal of disrupting NS/EP telecommunications could result in widespread outages.

3.0 RELATED THREAT ISSUES

The electronic intrusion threat to NS/EP telecommunications is not limited to members of the computer underground whose activities have been described in earlier chapters as "electronic intruders." Three other significant threats arise from telecommunication industry insiders, industrial espionage operations, and foreign organizations.

3.1 The Insider Threat

Telecommunication industry employees could constitute a threat to the integrity of the PSN if they engage in unauthorized activities. While there is little data on losses from attacks on PSN systems by telecommunications employees, data from the computer/data processing industry shows that between 60% and 80% of all malicious computer crime is committed by insiders.

The insider threat has three dimensions. The first type of insider threat is *honest mistakes or bad judgment calls* made by employees. While most studies show that this threat constitutes a large percentage of dollar losses, these losses are not caused by a malicious or unauthorized action, and are therefore not within the scope of this discussion.

The second type of insider threat is the *disgruntled employee*. These employees are motivated by revenge or spite against his/her employer. These persons' actions can range from spontaneous minor activities (such as deleting files in a computer) to highly orchestrated widespread attacks. These persons almost always seek to damage the company either physically or economically.⁴ The current economic climate within the telecommunications industry, given the trends toward downsizing and increased automation, has created potentially large numbers of disgruntled employees.

The third type of insider threat is the *coopted employee or mole*. These are normal employees who are approached by electronic intruders, saboteurs, foreign agents, or other criminals and offered money to gather information or provide access to systems. This threat can be highly insidious and result in large losses for companies because a mole may remain in his or her job for years without raising suspicion -- while providing a stream of highly sensitive data to outsiders. Frequently, disgruntled employees turn into moles if they perceive that they can harm their former organizations or if they can profit from these actions. In addition, a variety of other persons who have skills, knowledge, and access may be coopted by malicious outsiders; including vendors, contractors, customers, and employee family members. Dozens of moles have been discovered in the telecommunications industry in the past few years.

3.2 Industrial Espionage

Historically, industrial espionage has not proven to be a major activity in the computer underground. The reasons for this probably stem from the natural adversarial relationship between corporations and members of the computer underground. Corporations and the public perceive

⁴ Occasionally, disgruntled employees seek to harm specific persons (such as a former supervisor or coworker), rather than the company. These cases are outside the topic of this discussion.

electronic intruders to be unscrupulous, untrustworthy, and opportunistic. However, a new generation of electronic intruders emerging in the computer underground are motivated primarily by financial gain. These electronic intruders have offered their services for sale, and even published price-lists for their services. Electronic intruders' skills in communications monitoring, PSN data base access, and social engineering make their services highly valuable to industrial spies.

3.3 Foreign Involvement

The issue of foreign involvement in electronic intrusion activities in the United States PSN is complex. Telecommunications networks are truly international. They stretch beyond national boundaries, they bridge continents, and they provide connectivity to virtually every corner of the globe. Consequently, electronic intrusion activities are also international in scope. Many developed countries have computer underground movements that include toll fraud, virus creation, computer intrusion, and network attacks -- including the Netherlands, Canada, Brazil, Israel, Australia, Spain, Sweden, Ireland, United Kingdom, Czech Republic, Bulgaria, Russia, Austria, Greece, Switzerland, Romania, Hungary, Germany, Belgium, France, Italy, South Africa, Belarus, and Japan.

Over the past decade, networks in many other countries have been the target of intrusions by computer criminals. Since the world's telecommunications networks do not stop at national boundaries, electronic intruders regularly attempt to penetrate systems outside their own countries. In fact, most electronic intruders view "cyberspace" as a universe free from political boundaries. Given the truly international nature of the computer underground, electronic intruders generally do not care where targeted network elements and computers physically reside.

Because the U.S. has the largest and one of the most highly developed information infrastructures in the world, foreign electronic intruders have been targeting systems in the United States for many years. The best documented example is the case of the Hannover Hacker documented in Cliff Stoll's book "The Cuckoo's Egg." Another recent example appeared in a videotape made by Emmanuel Goldstein, editor of 2600 magazine. He filmed a team of Dutch electronic intruders breaking into U.S. military computers by exploiting poor security practices. In these two cases, the targets for the intrusions were mainframe computers, not PSN network elements. However, the skill sets exhibited by these groups of intruders could be directed at PSN network elements as easily as mainframe computer centers.

All research indicates that this issue of international "transborder" electronic intruders is a diversion from the real issue, which is espionage. Electronic intruders in the U.S. have similar capabilities and motivations to those in foreign countries. The one exception to this rule, however, is the involvement of foreign intelligence services (FIS).

Foreign intelligence services have directed activities against computer systems in the United States in recent years. The most well publicized of these cases involved *Pengo* and the *Chaos Computer Club* (CCC) in Germany. In this case, the KGB recruited German electronic intruders in the Hannover branch of the CCC, including Hans Hubner (alias Pengo). Pengo was actively engaged in electronic intrusion activity at the time (mid 1980s). He and several of his CCC colleagues acted at the direction of a KGB case officer and compromised several military and research computers in the U.S. In return, they received money, drugs, and other favors from the KGB.

The Hannover incident illustrates that many electronic intruders have developed advanced skill sets for gathering data electronically. As noted in Section 3, electronic intruders have also developed other skill sets directly related to telecommunications networks that focus on service disruption, denial of service, and fraud. These skill sets have always been high-interest targets for FISs. There has been little unclassified evidence that FISs have directly targeted, penetrated, or compromised the PSN in the U.S. However, there is a great deal of circumstantial evidence and speculation in the open source literature concerning foreign adversaries such as Libya, Iraq, and Iran targeting the U.S. networks. Many open source reports have postulated about the motivations of the former KGB and GRU intelligence services in Russia. National news organizations have also raised questions about the electronic espionage activities of FISs in "friendly" countries such as France, Japan, and Israel. However, to date, there has been little unclassified evidence of direct FIS actions against U.S. PSN assets.

This is not to say that a potential threat does not exist. A number of espionage opportunities requiring *data gathering* and *denial of service* skills have arisen recently for FISs. These require the services of electronic intruders or telecommunications employees -- both of which could be high payoff operations. The electronic intruder's recruitment and direction of activities would be much lower risk than the employee's. Electronic intruders can be enticed into attacking the PSN in a variety of ways: common interests, offering the technical support of intelligence officers, or merely challenging electronic intruders' technical capability. However, a disgruntled employee, once recruited based upon money or revenge, provides the ideal insider whose actions -- and results -- are more reliable.

4.0 POTENTIAL IMPACT ON NS/EP TELECOMMUNICATIONS

Sections two and three of this document outline the capabilities possessed by electronic intruders to disrupt and monitor telecommunications services. This section of the report seeks to describe the potential effects of these threats. If these threats were fully realized, significant effects would be felt by NS/EP telecommunications users.

More than 90% of Government telecommunications traffic transits leased PSN facilities. Consequently, the impacts of any security problem with the PSN has the potential to affect NS/EP users. However, targeting of specific Government telecommunications systems and services could more directly cause NS/EP telecommunications users to experience these effects. These effects are discussed in the paragraphs below.

4.1 Denial or Disruption of Service

Denial or disruption of service can be caused either intentionally or unintentionally by electronic intruders. Intentional disruptions have not been common in past years because most "smart" electronic intruders do not want to destroy the systems where they are working. However, this situation is changing. A new breed of electronic intruder is being motivated by financial gain and these individuals would undoubtedly disrupt PSN services if the price were right. In the past 3 years, electronic intruders have crashed or disrupted signal transfer points (STPs), switches, OAM&P systems, and other network elements.

The Government's position on the electronic intrusion threat, based on Department of Defense and Department of Justice input and analysis, identifies three key concerns:

"... denial of service, unauthorized monitoring, and remote points of origin external to the United States. These concerns are reflected in the capabilities of intruders that were noted in documented case studies of PSN intrusions."

During its deliberations in 1991, the President's National Security Telecommunications Advisory Committee (NSTAC) Network Security Task Force framed the denial of service issue in this manner:

"A motivated and resourceful adversary, in one concerted manipulation of network software, could degrade at least portions of the PSN and monitor or disrupt the telecommunications serving NS/EP users."

Many electronic intruders are highly skilled, knowledgeable individuals with engineering-level expertise in PSN systems. Adversaries would find these skills to be a high-interest item. Groups of electronic intruders, when organized and funded by interested adversaries, have the capabilities to launch sophisticated widespread attacks on the PSN. These types of attacks could result in significant degradations in the nation's NS/EP telecommunications capabilities, create significant public health and safety problems, and cause serious economic shocks.

4.2 Unauthorized Disclosure of Sensitive Information

Electronic intruders have demonstrated a high level of technical skills, and are able to capture information from the PSN and related systems in three primary ways:

- ***Electronic eavesdropping.*** Electronic intruders are able to monitor telecommunications circuits electronically, record telephone conversations remotely, capture and reproduce facsimile transmissions, and capture digital data off of monitored circuits. Frequently this digital data includes sensitive information such as login identifications, passwords, and source and target addresses.
- ***Packet data monitoring.*** Electronic intruders are able to electronically monitor packet data networks and reconstruct data streams using stolen or compromised X.25 diagnostics tools. This capability represents a significant improvement in previously reported electronic intruder capabilities involving PAD-to-PAD attacks.
- ***Electronically intruding on network elements.*** Electronic intruders are able to break into network elements that contain subscriber information such as names, addresses, cable pairs, and circuit termination points. They are able to electronically gather traffic and billing records and other sensitive NS/EP data. They are able to read and modify service classes, circuit identification numbers, and other codes associated with particular circuits.

The sheer volume of electronic intrusions on key network elements raises concern with the sensitivity of the information residing in network elements and data bases. While no known attacks have targeted network elements seeking to compromise large quantities of this data, in at least two instances NS/EP activities were severely compromised by electronic intruders.

4.3 Unauthorized Modification of Network Databases/Services

Electronic intruders have demonstrated a high level of technical skill in modifying PSN databases and subscriber services. They have added unauthorized accounts to service control points, service provisioning systems, digital cross connect systems, and other network elements. They have added and modified user services, forwarded calls, modified service classes on circuits, and turned off billing on specific circuits. On data networks, electronic intruders have changed the routing tables and service descriptions for specific users.

This level of penetration and skill illustrates the fact that electronic intruders could seriously compromise NS/EP telecommunications. An adversary would find these skills valuable in supporting intelligence gathering and espionage activities. Private citizens and corporations have been targeted by electronic intruders with these types of attacks. The likelihood of these types of attacks continues to increase because attacks of this type do not require large-scale technical resources to complete. Therefore, they are attractive to electronic intruders and other hostile adversaries

4.4 Targeting of Government Telecommunications Systems/Services

There are many types of NS/EP telecommunications systems and services which exist to fulfill specific NS/EP missions. Some are highly complex offerings, while others are little more than specialized commercial services established for Government use. One common thread unites virtually all of these NS/EP systems and services -- an overwhelming majority either transit or reside on existing PSN facilities. From the PSN's perspective, most NS/EP traffic is indistinguishable from normal traffic. Because of this reliance on the PSN infrastructure, most NS/EP systems and services are vulnerable to some or all the threats described in this document.

Five specific threats have the potential to affect NS/EP telecommunications services. These are discussed below:

- Some special Government services store their service access codes on network elements. The types of network elements storing these codes have experienced numerous unauthorized intrusions over the past 24 months. These intrusions were not targeted toward any specific Government NS/EP services.
- A special Government service provides emergency restoration and provisioning of telecommunications circuits. This service relies on specific priority codes to be included with each circuit's service records. These records are managed and maintained on network elements that have a long history of vulnerabilities from electronic intrusions.
- In several computer underground publications electronic intruders have discussed methods for exploring a dedicated Government NPA. Because of the lack of open source data on this NPA, electronic intruders have not made many inroads, but this will change over time. Electronic intruders clearly view the *mystery* of this NPA (and other dedicated Government telecommunications services) as a challenge.

- Electronic intruders have explored and compromised E911 systems. Significant degradations of service for E911 systems are possible if they were to be targeted by malicious electronic intruders.
- Systems which support DoD force projection are high-profile targets during military alerts and periods of national emergency. There have been many unconfirmed reports of U.S. military communications systems being targeted during recent military actions published in the open source literature. Even though these sources can not be confirmed in an unclassified context, military communications systems are an obvious target for espionage and counterforce activities by adversaries.

Any Government service which transits or resides on PSN facilities is vulnerable to the same sort of electronic intrusion threats faced by non-Government services. Threats from service disruption, denial of service, unauthorized disclosure of data, unauthorized modification of service, and fraud are present in the PSN and should be considered when making contingency and emergency service plans.

APPENDIX A
REFERENCES

APPENDIX A REFERENCES

2600SP93

Spring 1993. "Letters to the Editor." *2600: The Hacker Quarterly*. Volume 10. Number 1.

2600SU93

Summer 1993. "A Guide to the 5ESS." *2600: The Hacker Quarterly*. Volume 10. Number 2.

2600VID

Autumn 1991. "The Hacker Video." *2600: The Hacker Quarterly*. Volume 8. Number 3.

ACPREFIX

February 1992. Internet newsgroup discussion on 800 database.

ALEXANDER91

Alexander, Michael. October 21, 1991. "Justice Unit Spurred on by Cross-border Hackers." *Computerworld*.

AP4989

April 9, 1989. "Crackdown on Hackers Urged." Associated Press.

AP51388

May 13, 1988. "Virus Hits UNIX at Bell Labs." Associated Press.

ASSIST103

1991. "Security Alert for Novell Network Software." ASSIST Bulletin 91-3.

BARLOW90

Barlow, John. 1990. "Crime and Puzzlement, Parts 1 and 2." *Whole Earth Review* and other computer underground publications.

BASNET1

Sk8 The SkinHead (hacker alias). November 1989 (est.). "Basic Networking."

BELLTRASH

The Dragyn (hacker alias). 1989 (est.). "Bell Trashing."

BOOTLEG6

June 1992. Reprint of "Cracking Down on Abuse." *MCI World*.

BRUNNER75

Brunner, John. 1975. *The Shockwave Rider*. New York: Ballantine.

BRUNNSTEIN91

Brunnstein, Klaus. July 29, 1991. *Computerworld*.

BULLIES

Flanagan, William G. December 1992. "The Playground Bullies Are Learning How to Type." *Forbes*.

CAPITAL92

Tichy, Roland. October, 1992. "Authentic, Topical, Inexpensive." *Capital*.

CC593A

Rothfeder, Jeffrey. May 1993. "Holes in the Net." *Corporate Computing*.

CC593B

Quinn, Brian. May 1993. "Dialing for Dollars." *Corporate Computing*.

CDUGD91

February 1991. *Computer Down-Under-Ground Digest, Issue 1*.

CFCA91

Communications Fraud Control Association. 1991. *Annual Fraud Estimates*.

CFCA193

Communications Fraud Control Association. December 1992 - January 1993. *Communicator*.

CFSB791

Klopp, Charlotte. July 1991. *Computer Fraud & Security Bulletin*.

CHENOWITH92

Chenowith, Richard. 1992. "Allpoints." *Intercon Security, Ltd*.

CIA92

1992 (est.). *Criminals Into Anarchy*. Volume 1, Issue 1.

COMPCONF91

November 21, 1991. "Prosecution & Defense." *The Computer Conference Newsletter*.

COOK90

Cook, W.J. May 1990. "Uncovering the Mystery of Shadowhawk." *Security Management*.

COSMOS86

Sir William (hacker alias). 1986. "The 1986 COSMOS Files Part III: Service Order Input."

CPP92

The Raven (hacker alias). 1992. "The Ultimate Cellular Phone Phreaking Manual, Parts 1 and 2."

CRIMCOST

Kay Russell. "Calculating the Cost of Computer Crime." *Infosecurity News*, Volume 3, Number 6.

CUD104 through CUD530

April 11, 1990 to April 24, 1993. *Computer Underground Digest*. Volume 1, Issue 4 through Volume 5, Issue 30.

CW52592

Schwartz, Jeffrey. May 25, 1992. "Users Size Up Hacker Tracker." *Communications Week*.

DATAPRO

August 1992. "Common Channel Signaling System Number 7." Datapro Information Services Group.

DEBATE

McMullen, Barbara and John McMullen. March 18, 1991. "Hacker' Debate Heats Virus Conference." *Newsbytes*.

DENNING90

Denning, Dorothy. 1990. "Concerning Hackers Who Break Into Computer Systems." DEC Systems Research Center.

DFP1 through DFP4

January 1992 to May 1992. *Digital Free Press*. Volume 1 through Volume 4.

DIA90

Young, Stanley and Michael Higgins. June 14, 1990. "Threat to the Public Switched Network." Presentation to the Defense Intelligence Agency.

DIA93

May 10, 1993. Position statement of the U.S. Government on electronic PSN intrusions. Based on December 3, 1990, briefing to NSTAC.

DIGITAL

Phantasm (hacker alias). September 12, 1992. "Digital Underground."

DM11091

October 1991. *Digital Murder*. Volume 1.

EDWARDS92

Edwards, Dr. Jack. March 1992. Meeting of the Network Security Task Force.

EFF402

December 1992. *EFFector Online*. Volume 4, Issue 2.

FRAUDSEC

Stusser, Daniel. "Securing Your Systems Against Fraud." *Networking Management*. Volume 10, Issue 6.

FREEDMAN93

Freedman, Alan. 1993. *Electronic Computer Glossary*. Electronic publication.

GE5

Hayduke, George. 1989. *The Get Even Series*. Port Townsend, WA: Loompanics Unlimited.

GIBSON84

Gibson, William. 1984. *Neuromancer*. New York: Ace.

GREEN92

Green, James Harry. 1992. *The Business One Irwin Handbook of Telecommunications*. 2nd ed. Homewood, Illinois: Business One Irwin.

HACKDEA

Bushaus, Dawn. December 1990. "DEA Falls Prey to Hackers; Theft of Services Could Amount to \$2 Million." *Communications Week*.

HACKGUIDE

The Mentor (hacker alias). December 1988. *A Novice's Guide to Hacking - 1989 Edition*.

HACKUNLM

October 1989. *Hackers Unlimited Magazine*. Volume 1. Issue 1.

HAFFNER91

Haffner, Katie and John Markoff. 1991. *Cyberpunk: Outlaws and Hackers on the Computer Frontier*. New York: Simon and Schuster.

HD07

Hacker Supreme (hacker alias). 1986. *Hackers' Directory*. Volume 7.

IHA191

June 1991. *International Hackers Association Newsletter*. Volume 1.

INDUSTRIAL1192

November 1992. "Competitive Intelligence; A Key to Marketplace Survival." *Industrial Marketing*.

INFORM2

January 1992. *Informatik*. Issue 2.

JROGR149

Jolly Roger (hacker alias). 1990 (est.). "A Short History of Phreaking."

LANDRETH85

Landreth, Bill. 1985. *Out of the Inner Circle: A Hacker's Guide to Computer Security*. Bellevue, Washington: Microsoft Press.

LANMAG93

June 1993. "Glossary." *LAN Magazine*. Volume 8. Number 6.

LEVY84

Levy, Steven. 1984. *Hackers: Heroes of the Computer Revolution*. Garden City, Long Island: Doubleday.

LOD1 through LOD4

January 1, 1987 to May 20, 1990. *Legion of Doom/Legion of Hackers: Technical Journal*. Volume 1 through Volume 4.

LODINDICT90

U.S.A. vs. Robert Tiggs and Craig Neidorf. U.S. District Court, Northern District of Illinois, Eastern Division. No. 90, CR 70. Violations: Title 18, United States Code, Sections 1343 and 2314. 1990.

LOL012

October 22, 1991. *Legion of Lucifer - Phone Hackers United to Crash & Kill Newsletter*. Volume 1. Issue 12.

LOL020

August 29, 1991. *Legion of Lucifer - Phone Hackers United to Crash & Kill Newsletter*. Volume 1. Issue 20.

LT42393

April 23, 1993. "Computer Hacker Accused of Unfairly Winning Prizes." *Los Angeles Times*.

MARTEN76

Marten, James. 1976. *Telecommunications and the Computer*. Englewood Cliffs, NJ: Prentice-Hall, Inc.

MEYER89

Meyer, Gordon R. 1989. "The Social Organization of the Computer Underground." M.A. Thesis, Northern Illinois University, De Kalb.

MEYERTHOMAS90

Meyer, Gordon R. and Jim Thomas. 1990. "The Baudy World of the Byte Bandit: A Postmodernist Interpretation of the Computer Underground." Department of Sociology, Northern Illinois University.

MITNICK4

Kellner, Mark. January 2, 1989. "Hacker is Jailed for Theft: Allegedly Steals DEC VMS Code." *MIS Week*.

MTRASH

Kid & Co. and The Shadow (hacker aliases). 1984. "More on Trashing." *2600: The Hacker Quarterly*.

NB12090

Woods, Wendy. January 20, 1990. "Three Indicted for Stealing Classified Data." *Newsbytes*.

NCSA5692

National Computer Security Association. May/June 1992. *NCSA News*.

NCS-M93

Manager, National Communications System. August 1993. "Status Report on the Security of the Public Switched Network: Report to the Chairman, Inter-Agency Working Group."

NEWTON93

Newton, Harry. 1993. *Newton's Telecom Dictionary*. Electronic publication.

NFX5

October 1991. *New Fone Express*. Issue 5.

NFX61192

November 1991. *New Fone eXpress*. Volume 6.

NRC89

National Research Council. 1989. "Growing Vulnerability of the Public Switched Network: Implications for National Security Emergency Preparedness."

NSA102

June 23, 1991. *National Security Anarchists*. Volume 1. Issue 2.

NSA103

July 1, 1991. *National Security Anarchists*. Volume 1. Issue 3.

NSTAC90

National Security Telecommunications Advisory Committee. 1990. "Proceedings of the Network Security Task Force."

NSTAC92

National Security Telecommunications Advisory Committee. 1992. "Proceedings of the Network Security Task Force."

NSTF90

November 1990. Network Security Task Force Report to NSTAC XII.

NSTF92

June 1992. Network Security Task Force Report to NSTAC XIV.

NW6192

Taff, Anita. June 1, 1992. "Bill Would Protect Government Agencies from Toll Fraud Charges." *Network World*.

OTA87

U.S Congress, Office of Technology Assessment. October 1987. *Defending Secrets, Sharing Data: New Locks and Keys for Electronic Information, OTA-CIT-310*. Washington, DC: U.S. Government Printing.

PARKER83

Parker, Donn. 1983. *Fighting Computer Crime*. New York: Charles Scribner's Sons.

PHANTSY10

November 1, 1992. *Phantasy*. Volume 3. Issue 10.

PHANTSY11

November 6, 1992. *Phantasy*. Volume 3. Issue 11.

PHRACK01 through PHRACK43

November 1985 to July 1993. *PHRACK Magazine*. Volume 1 through Volume 43.

PHN02-04

1989 (est.). *P/HUN Newsletter*. Volume 2. Issue 4.

PHUN88

September 30, 1988. *P/HUN Newsletter*. Volume 1.

POST32391

Potts, Mark. March 23, 1991. "Hacker Pleads Guilty in AT&T Case." *Washington Post*.

POWELL90

Powell, Dave. September 1990. "Network Abuse: Who's the Enemy." *Networking Management*.

QUARTERMAN90

Quarterman, John S. 1990. *The Matrix: Computer Networks and Conferencing Systems Worldwide*. Bedford, Massachusetts: Digital Press.

R&ROP

Fred Steinbeck (hacker alias). 1991 (est.). "Dealing with the Rate and Route Operator."

RAYMOND91

Raymond, Eric. 1991. *The New Hacker's Dictionary*. London: The MIT Press.

RSKS1364

July 14, 1992. *RISKS Digest*. Volume 13. Issue 64.

RSKS1438

March 7, 1993. *RISKS Digest*. Volume 14. Issue 38.

SANDZA84

Sandza, Richard. November 12, 1984. "Night of the Hackers." *Newsweek*.

SCHWEIZER93

Schweizer, Peter. 1993. *Friendly Spies: How America's Allies Are Using Economic Espionage to Steal Our Secrets*. New York: The Atlantic Monthly Press.

SECMAN1-93

January 1993. *Security Management Magazine*.

SHERMAN85

Sherman, Kenneth. 1985. *Data Communications: A User's Guide*. Reston, Virginia: Reston Publishing Company, Inc.

SJMN41391

Barnum, Alex. April 13, 1991. "Computer Fugitive Fits In: Man Eluded FBI With Low Profile." *San Jose Mercury News*.

SJMN52791

Barnum, Alex. May 27, 1991. "Hacker's Obsession Led Him to Jail." *San Jose Mercury News*.

SOCENG89

Fallen Angel (hacker alias). September 1989. "Social Engineering: How to Get Information."

SPOOFER91

Goodfellow, Geoffrey S., Robert N. Jesse, and Andrew H. Lamothe, Jr. 1991. "The Electronic Serial Number: A Cellular Sieve? Spoofer Can Defraud Users and Carriers."

SRI93

SRI International. 1993. "Vulnerabilities of the PSN."

STOLL89

Stoll, Clifford. 1989. *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. New York: Doubleday.

STOLL89-2

Stoll, Clifford. May 1988. "Stalking the Wily Hacker." *Communications of the ACM*.

STOLL89-3

Stoll, Clifford. September 1987. "What Do You Feed a Trojan Horse?" Proceedings of the 10th National Computer Security Conference.

SWEDISH90

1990. *Annual Year Protocol of the Swedish Hackers Association*. Volume 3.

SWEDISH92

February 1992. *Annual Year Protocol of the Swedish Hackers Association*. Volume 4.

TAOTRASH

The Phoenix Force (hacker group). 1988 (est.). "Art of Trashing."

TD1190

November 1990. *Telecom Digest Guide to Special Prefixes/Numbers*.

THEFT

Grata, Joe. August 29, 1993. "2 Teen 'Hackers' Held in Break-ins Troopers Seek 4 other 'Computer Whizzes' in Equipment Thefts." *Pittsburgh Post-Gazette*.

TNS10

November 18, 1987. *Tolmes News Service*. Volume 10.

TRASHTECH

Master of Reality (hacker alias). 1989 (est.). "Trashing Techniques."

UKACT05

Gold, Steve. May 8, 1990. "UK Anti-Hacking Bill Goes Through to House of Lords." *Newsbytes*.

UKACT07

Gold, Steve. July 5, 1990. "UK: Computer Misuse Bill receives royal assent." *Newsbytes*.

UMPOULSEN

Episode featuring the case of Kevin Poulsen. New York: NBC television program, *Unsolved Mysteries*, May 1991.

UNLISTED

The Jolly Roger (hacker alias). 1990 (est.). "Unlisted Phone Numbers."

USRESEARCH

USA Research. 1992. "1992 IPA Computer Virus and Hacker Study."

UXU002

1991. *Underground eXperts United*. Volume 1. Issue 2

UXU033

1991. *Underground eXperts United*. File 33.

UXU047192

January 1992. *Underground eXperts United*. Volume 47.

WINTERMUTE1

Wintermute (hacker alias). February 1991. Unnamed article on Cyberpunk.

WSJ082290

Wilke, John R. August 22, 1990. "Open Sesame: In the Arcane Culture of Computer Hackers, Few Doors Stay Closed." *Wall Street Journal*.

ZONE2

Black Death (hacker alias). 1991 (est.). "The Phreaking Articles - Volume 2."