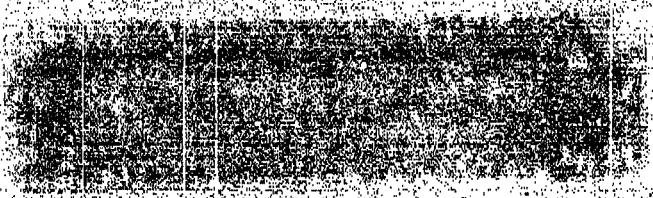
Proceedings of the 8th NATIONAL COMPUTER SECURITY CONFERENCE



30 SEPTEMBER-3 OCTOBER 1985

GAITHERSBURG, MD.

PREFACE

This is the eighth in a series of conferences co-sponsored by the DoD Computer Security Center and the National Bureau of Standards. The theme of this year's conference is, "Computer Security in the National Arena." The program is directed toward the users as well as the developers of computer security products, and includes presentations on the efforts of the Department of Defense and National Bureau of Standards, of trusted product researchers and developers, and of the private sector. The specific topics this year include secure networks, verification, assurance, formal models, security architecture, sanitization, applications on secure bases, labeling, a profile of hackers, performance tuning of secure products, and market analysis of secure products.

In January, 1981, the Computer Security Center was established in the Department of Defense. The Center is encouraging the development of trusted computing systems through technology transfer with industry, and is defining ADP system evaluation procedures to be applied to both Government-developed and commercially developed trusted computing systems.

In September, 1984, the President signed National Security Decision Directive 145, which enlarged the mission of the Center to include the Federal civil establishment and some segments of the private sector.

The National Bureau of Standards' Institute for Computer Sciences and Technology, through its Computer Security and Risk Management Standards Program, seeks new technology to satisfy Federal ADP security requirements. The Institute then promulgates cost-effective technology in Federal Information Processing Standards and Guidelines. The Institute is cooperating with the Department of Defense in transferring the results of the Computer Security Center's research throughout the Government and to private industry.

TABLE OF CONTENTS

WELCOMING REMARKS

Robert L. Brotzman

Director, DoD Computer Security Center
Fort George G. Meade, Maryland 20755-6000

I'd like to join Jim Burrows in welcoming you to the 8th National Computer Security Conference, my second such conference here at the National Bureau of Standards.

Since I last spoke to you a year ago, the Center has grown 68 percent from 160 personnel to our present staff of about 270. Much of the Center's rapid personnel growth is due to the need to staff a very young organization that acquired national responsibilities for computer security when President Reagan signed National Security Decision Directive 145 (NSDD-145) last September. The Center has broadened its mission from the defense level to the national level. By expanding the scope of our mission, we have changed the nature of our clientele to include civil government departments and agencies and have redirected the emphasis of our activities. Before I relate to you how we have restructured our computer security program, I would like to outline a few of the assumptions we made about the nature of our new mission and the conditions under which it would be performed.

We believe that most of the civil agencies are less aware of their computer security needs than are the agencies and departments within the Department of Defense. To the extent that they do recognize those needs, they apparently perceive them as quite different from those within Defense. To service this new community, we have to raise their awareness, gain an understanding of their computer security concerns, and determine which of their needs are truly unique and which have common threads. We do not have all the answers for the problems of the civil sector of government; if we did, those answers could be effectively applied only if the people concerned recognized them as valid. There will be no salvation in a forced conversion.

The inherent ability of current computer systems to protect themselves and their data from abuse is appallingly low. Treatment is available right now which can improve their ability to function securely, but nursing systems that were born weak is only a stop-gap, not a solution. To reduce these risks substantially, we need to develop and apply methodology that will enable us to create systems with solid security features designed in from the beginning. The strategy we are pursuing consists of a three-pronged attack; firstly, we are encouraging the use of software packages which reduce the risk to many current systems. Secondly, we are encouraging industry to develop systems defined in the Department of Defense Trusted Computer System Evaluation Criteria. Thirdly, we are embarking upon an aggressive research program to substantially improve the state of knowledge about computer security. In general, this strategy attempts to do immediately, those things which we know how to do, and to proceed in parallel to try to improve the medium-term security posture and start the research program needed to learn far more about the subject.

To raise awareness to the computer security problem, we have provided speakers to numerous conferences and courses. Two computer security awareness films produced by the Center have been released nationally. Within three days of announcing the availability of the films in "Government Computer News," we were contacted by more than 100 agencies and organizations seeking copies of the films. Over the next few years, we expect to produce several films and video tapes on subjects such as password management and the Trusted Computer Systems Evaluation Criteria.

The initial steps have been taken toward developing and implementing an education and awareness program. Through this program we expect to train or influence a massive audience, primarily (but not exclusively) within the government, to be aware of the need for computer security and of the techniques for maintaining it. Security literatrue and posters will be distributed, "awareness exhibits" will be provided at symposiums, and a variety of awareness training will be given. A National Training Plan has been drafted to address computer security on a national level. Some existing computer security courses are being modified to remove classified material so that they may be given to a wider audience. This annual conference has been refocused to better include the civil government and private sector audiences.

To gain a better understanding of our new clientele, we are initiating an education needs assessment program for the federal government. To extend our knowledge of security problems beyond government circles, an effort has begun to study computer abuse in the private sector. We have established informal contacts with agencies such as the Department of Energy, the Department of Commerce, the Federal Bureau of Investigation, and the General Services Administration. A survey of Defense Department contractors' use of hardware and software, similar to that done for the Department of Defense, is now underway. We have responded positively to requests for assistance from the Federal Emergency Management Agency, the Forestry Service, and the Treasury. In providing

these agencies with the assistance they requested, we are gaining a greater understanding of their needs.

Within the Center, we plan to build an automated data processing operation that can serve as a showcase for the departments and agencies of government so that they can see how security features can be applied in today's operations.

A new division has been created to interact with the commercial world and to encourage vendors to consider security in the design phase of all their products. In March of this year, we sponsored a conference for computer vendors which was attended by about two hundred people from industry. We intend to make this an annual event. We have met several times with the Computer Business Equipment Manufacturer's Association and a promising relationship is being developed with that body.

While to date it has only been published as a Computer Security Center document, the Trusted Computer System Evaluation Criteria (commonly known as the "Orange Book") is widely accepted as the authoritative document in that area. About 22,000 copies have already been distributed. Coordination is nearly completed to make it a Department of Defense standard. Now that our mission is extended, efforts are underway to determine the revisions needed to make the "Orange Book" more applicable to the civil sector.

An "environmental guideline" document that provides guidance on how to use the "Orange Book" is about to be published as a Computer Security Center document. It too will be revised to meet the needs of those outside the Defense Department. We have also begun to develop Audit Guidelines, Discretionary Access Control Guidelines, Network Security Criteria, Data Base Management Criteria, and Office Automation Security Guidelines.

The threat imposed by insecure systems is one that has to be addressed. It isn't just a Department of Defense problem, although the threat to our security is paramount. It isn't just a governmental problem. It looms over virtually every aspect of our lives, collectively and individually -- public and private. There are no panaceas, but there are steps that can be taken to reduce the danger. We need to take these steps as quickly as our capabilities allow. Whatever can be done now, should be done now. Whatever is postponed will be harder to do and probably more expensive in the future, but what's worse, postponement means a longer time of having to depend upon systems that are all too vulnerable.

The structure created by the NSDD 145 has raised the level and broadened the scope of the attention given to computer security. Within the Computer Security Center we have charted a course which we believe to be responsive to the task set before us. We are aggressively pursuing initiatives that address both current and future needs. The actions we are taking come with no guarantees; failure to act does carry a guarantee - that of a major disaster.

Protecting the nation's information base and the equipment which processes that information is a monumental task. If we can do it and do it well, we will survive as an information-based open society. That is well worth doing.

KEYNOTE ADDRESS--COMPUTER SECURITY FOR THE NATION

Lieutenant General William E. Odom, USA,
Director, National Security Agency
Chief, Central Security Service

I am pleased to have the opportunity to address the 8th National Computer Security Conference cosponsored by the Department of Defense Computer Security Center and the National Bureau of Standards Institute for Computer Sciences and Technology. I am equally pleased to see the high level of interest in this forum. I take it as indicative of a heightened perception of the vulnerabilities of our computer assets and a commitment to do something about it.

In today's world the prime force for change is the computer. It is reshaping our lives in the second half of the century, just as the automobile and plane did in the first half, and industrialization did in the second half of the nineteenth century. But as the old adage says, there is no rose without a thorn. Our initial wonderment with the beauty has blinded us to the thorns. As with other innovations, we have had a difficulty bringing ourselves to deal with the problems until well after they have become entrenched in our daily lives.

For many decades we accepted the convenience of the automobile as an unmixed joy. Then one day we awoke to find a polluted atmosphere, 50,000 auto-related deaths per year, a public transportation system in shambles, and a dependence on foreign oil that imperiled our economy and our security. We are now confronting these problems at great cost and with limited success. None of these problems arose overnight; and each could have been dealt with more easily, cheaply and successfully had we the foresight to recognize and the will to solve them earlier.

We cannot retreat from this new revolution, the computer revolution, nor would we want to. It serves us well. The computer is assuming our burdens on tasks that are tedious, and giving us the power to break through barriers that once seemed insurmountable. Along with other technology, it is expanding the forms, speed, reliability, and extent of our communications systems.

Over the past twenty years the subject of computer security has emerged from the world of the computer specialist into the national arena. Computer security efforts in the distant past -- the late 1960's to the mid 1970's --were relatively small. Nevertheless, achievement ensued. Researchers phrased the right questions, isolated the facts, and debated the relative merits of various technical approaches. They then proceeded to test many of the approaches, and learned from both the successes and the failures. Some of the successes led to the current arsenal of trusted computer products. Although the middle years, from the mid 1970's to the early 1980's, showed a reduced emphasis and effort, there is emerging a renewed interest in and support for computer security. Within the last year or two, a general consciousness raising has spread across the nation. Media coverage of computer crime and youthful intruders has helped to raise national computer security awareness. Such crimes as electronic funds transfer theft, welfare fraud, and the misuse and alteration of private credit records have been widely publicized. Moreover, in the past six months, articles on the DoD Computer Security Center and its national responsibilities for computer security have appeared in magazines such as NEWSWEEK and DATAMATION.

Recognition of the nation's need for computer security has definitely made its way to the top. Last fall at about this time, President Reagan recognized the severity of the vulnerabilities in the nation's computer systems by signing the National Policy on Telecommunications and Automated Information Systems Security, otherwise known as National Security Decision Directive Number 145 (NSDD-145). This directive is the result of years of effort to give national attention to the protection of information involving these two advanced technologies. It is not my intention to recite the full catalog of provisions from this Directive. I do want to point out, however, that the creation of a national committee structure to address the information security concerns of the Federal Government and of the country is a powerful recognition of the broad and important problem that we face.

NSDD-145 introduces several significant changes to the way we have been doing business. Foremost, this directive recognizes the increased merging of telecommunications and automated information systems and their interdependence. It also prescribes reorganization and refocusing of the national communications security and computer security objectives, policies, and organizational structure. In short, it provides central leadership for computer security which was previously non-existent.

In addition to government classified information, NSDD-145 broadens the information protection policy to include sensitive, but unclassified, government

or government-derived information and sensitive non-government information. "Sensitive" implies that the loss of such information could adversely affect national security. With classified information, the systems are secured as necessary to prevent compromise or exploitation. With regard to other sensitive information, the protection shall be in proportion to the threat and potential damage to the national security. This policy means that our responsibility for information protection extends across the entire Federal Government and in some instances requires the cooperation of the private sector.

However, recognizing a problem is far easier than developing ways to bridge organizational and philosophical divisions of responsibility. I'm referring specifically to the Directive's requirement that "the government shall encourage, advise, and, where appropriate, assist the private sector to: identify systems which handle sensitive non-government information, the loss of which could adversely affect the national security." The government would also assist the private sector in determining the vulnerability of these systems and formulate strategies for providing protection in proportion to the threat of exploitation.

We realize the private sector's apprehension regarding NSDD-145, illustrated well by an editorial cartoon in "COMPUTERWORLD". In the cartoon, entitled, "Snakes in the Woodpile," three snakes labelled the National Security Agency, the Department of Defense, and the National Security Council, are intertwined under logs representing NSDD-145. The caption underneath reads "Private sector security control." The Federal Government in no way wants to assume the "big brother" role with private industry. Instead, it will actively seek information and advice from the private sector with regard to this policy.

The NSDD-145 also establishes an infrastructure for carrying out this more encompassing approach to automated information systems security. Significantly, it expands the current authority of the Secretary of Defense as the Executive Agent of the Government for Communications Security to include Automated Information Systems Security. As the operational arm of the Executive Agent, the Director of the National Security Agency, has become the National Manager for Telecommunications and Automated Information Systems Security. The Executive Agent and the National Manager respond to direction from the Systems Security Steering Group--a group with Cabinet-level membership under the National Security Council (NSC). The Director, NSA, also serves as Executive Secretary to this Steering Group. This group replaces the NSC Subcommittee on Telecommunications Protection (established by Presidential Directive Number 24) and is expected to perform a more active review and oversight role.

The Steering Group will use the National Telecommunications and Information Systems Security Committee (NTISSC) to formulate operational policy, set objectives, and establish priorities. The NTISSC replaces and subsumes the National COMSEC Committee established in 1979. The 22 members of the NTISSC represent a broader participation by the military and the civil agencies. Because the members will be decision makers for their respective agencies and departments, the NTISSC has the potential to be a strong and unified force for coping with the overall information security challenge. The Heads of the Federal departments and agencies still have the responsibility for developing plans and programs for implementing the national policy within their organizations.

The DoD Computer Security Center, as Bob Brotzman has just told you, is also being redirected to continue computer security support on the national level. The Center was established in 1981 to support the Director NSA, as the DoD technical focus for computer security. Now the Center will be responsive to the needs of the entire Federal Government and its more than 1,000 departments, agencies, boards and commissions.

There are two subcommittees under the NTISSC --one for telecommunications and the other for automated information systems security. Under a requirement of the NSDD-145, the Subcommittee on Automated Information Systems Security (SAISS) has already submitted its first annual evaluation report to the NTISSC on the status of automated information systems security in the Federal Government. They found the current approach by the Federal Government was fragmented and sometimes inconsistent with regard to policy, procedures, tools, and mechanisms for protecting automated information systems resources. They also found that the lack of a clear policy mandating the use of technical protection measures and trusted computer systems does little to convince industry to respond to the government's computer security needs. As a result, even those security enhancements that are possible today are still not widely available. This situation is likely to continue unless there is an increase in awareness especially on the part of management, of the vulnerabilities and security risks associated with the use of automated information systems and networks. The SAISS is also developing standards on password usage and environment guidelines for the DoD Trusted Computer System Evaluation Criteria (commonly known as

the Orange Book); developing security criteria for personal and shared computers; and drafting a program to encourage advise, and assist the private sector in developing computer security products.

There have been some positive actions which can provide a foundation for building a sounder security future. This approach is being recommended by the SAISS and is representative of the thrust being pursued by the Federal Government. Some of the major undertakings include: (1) achieving at least the minimal level of computer security by implementing the technology and mechanisms that are available today; (2) fostering a greater awareness of the security risks and needs for effective computer security measures; (3) establishing consistent and meaningful computer security requirements and mandating that these be met; (4) exploring the possible consolidation of multiple policies into a responsive, uniform and comprehensive national policy framework with enforcement power; and (5) joining forces with industry not only to achieve near term computer security products, but to undertake a more aggressive R&D program for meeting the future technical needs.

To expand on the last point, the R&D challenge we face is an incredibly difficult one. We are really far down on the power curve. Compared to communications security where we well understand the problem and our challenge is to reproduce the solution efficiently and economically, the R&D challenge is still being defined. At this time we know the bounds of the computer security problem, but we have much work ahead in filling in the fine line details. With Bob Brotzman and Col Joe Greene leading the charge, we have undertaken a vigorous campaign to secure the necessary resources to get the job done. We have only been partially successful in the DoD resource arena, but we will keep the pressure on through all phases of the budget cycle. We believe, however, that we have secure enough funding to begin to prime the pump, and as Joe Greene will explain in the next presentation, we have a plan to move out and impact the industry and market place.

Since the issuance of NSDD-145 in September 1984, the participants have begun the process of making real and effective the mandate to protect the sensitive and classified information of this nation. No one expected the job to be easy, and it hasn't been. Few expected quick unanimity on the pressing problems and the solutions to those problems. Again experience has shown that diversity of opinion is likely to remain a dominant feature of computer security efforts for some time. But, it is from the plurality of viewpoints and needs -- tested and adjusted in an open forum --that the sorely needed advances in information security will come.

You, the practitioners, managers, and consumers of security technologies and products contribute to the national needs by bringing your requirements, your unique perspectives and ideas, and even your criticisms to forums like this one. I applaud your interest in being here. I picture each of you as representing scores of colleagues who will benefit by your presence and by the work being reported and reviewed here. I hope that you will spread your experiences here to bring benefits and continuing awareness to the nation.

LTG WILLIAM E. ODOM

LTG William E. Odom, USA, became the National Security Agency's 11th Director on May 8, 1985. General Odom's previous assignment, which he held since November 1981, was Assistant Chief of Staff for Intelligence, U.S. Army. He is the first army officer to head the Agency since LTG Marshall S. Carter who was director from 1965 to 1969.

A specialist in Soviet affairs, General Odom speaks Russian and reads German. He is a member of the Council on Foreign Relations, the American Political Science Association, the International Institute for Strategic Studies, and the American Association for the Advancement of Slavic Studies.

General Odom is the author of one book, **The Soviet Volunteers: Modernization and Bureaucracy in a Public Mass Organization**, and has written numerous articles about the Soviet Union for scholarly journals.

Before coming to the National Security Agency, General Odom served as Assistant Army Attache, Moscow; Associate Professor and Research Officer Department of Social Sciences, West Point; Military Assistant to the Assistant to the President for National Security Affairs; and Deputy Assistant Chief of Staff for Intelligence, U.S. Army.

General Odom graduated from West Point with a B.S. in Engineering. He also holds degrees from Columbia University: an M.A. in Russian Area Studies and a Ph.D. in Political Science. In addition, General Odom has attended The Infantry School (Basic Course), The Armor School (Advanced Course), and the U.S. Army Command and General Staff College.

DoD OVERVIEW
COMPUTER SECURITY PROGRAM DIRECTION

Colonel Joseph S. Greene, Jr., USAF

Deputy Director, DoD Computer Security Center
Fort George G. Meade, Maryland 20755-6000

BACKGROUND

At the 7th DoD/NBS Computer Security Conference (1984), and at the IEEE Computer Security Conference this spring, a number of people expressed interest in a review of new directions for the Department of Defense (DoD) Computer Security Program (CSP). This paper responds to those interests.

The DoD Computer Security Evaluation Center (DoDCSEC) was established in 1981. DoD Directive 5215.1 assigns responsibility for computer security and provides direction for the formulation of the Consolidated Computer Security Program. The Services and the Defense Communication Agency participate through the Technical Review Group in the formulation and execution of the CSP. The Director, National Security Agency, provides central oversight and single-point accountability for the CSP. The CSP funds the operation of the DoDCSEC and the generic Research, Development, Test, & Evaluation (RDT&E) program for the Department. Generic computer security research has potential application over a very broad, generalized basis and includes experimental exploration and development of feasible and potentially useful technology that is responsive to a broad class of computer security needs. Generic computer security research is distinct from application-dependent research and development for specific DoD component systems.

TECHNOLOGY BASE

In the past, the Department has responded to security needs by including computer security requirements in selected major programs (e.g., Strategic Air Command Digital Information Network (SACDIN), Automated Digital Information Network (AUTODIN), Defense Data Network (DDN), Inter-Service Agency Automated Message Processing Exchange (I-S/A AMPE), World Wide Military Command & Information Control System (WIS)). This approach tended to work for system-high systems but became increasingly expensive for multilevel secure trusted computer systems. With the trend toward ever more pervasive use and interconnection of Automated Information Systems (AIS), the case-by-case approach becomes prohibitively expensive. Without equivalent trust for all components, interconnected systems are only as secure as the weakest component, as can be demonstrated using password grabber, garbage collection, spoofing, and Trojan Horse attacks against a multilevel secure computer with an untrusted terminal. To encourage industry to incorporate trusted computer base security features and market trusted computers as their standard commercial offerings, the Department developed a strategy of publishing standards. encouraging industry to build trusted products, evaluating and certifying these products against the standard, and publishing the results as an Evaluated Products List (EPL). Through this strategy, the Department hopes to make trusted products available to all users and to spread the development cost over a larger segment of the industry.

Discussion of the CSP should be based on a common understanding of the several factors that define the current technology maturity of the trusted computer systems. The following treatment assumes that the reader is familiar with the basic features and assurances defined in the DoD Trusted Computer Security Evaluated Criteria[1] and the fundamental conclusion of penetration studies in the 1970's that computer security must be an inherent quality of the design and implementation of the computer. The aspects discussed next provide a partial summary of the trusted computer system technology base. Correcting and overcoming deficiency in the several areas identified will constitute part of the challenge of the CSP.

a. A recent survey of 17,070 DoD computers indicates that half should be upgraded with Discretionay Access Control (DAC) capabilities. Although about 30 major vendors, each with numerous machine/-operating system combinations, were identified in the survey, only three DAC packages have been certified and placed on the DoDCSEC Evaluated Products List. These are the International Business Machines, Corp. (IBM), Resource Access Control Facility (RACF); SKK Inc's, The Access Control Facility 2 (ACF2); and CGA, Software Products Group's, TOP SECRET, all for the IBM MVS operating system that accounts for less than 400 of the machines identified in the DoD Survey. In addition to insufficient coverage, the government is somewhat behind the private sector in employment of those security measures that are available. For example, of the 10,000 MVS licenses in the private sector, half use add-on security packages, while only about 40 percent of the 500-plus government-owned MVS systems are similarly protected. If all DoD IBM mainframes used MVS and had add-on security packages, only about 4 percent of the DoD systems needing DAC capabilities would be protected.

b. According to the May 1985 Five-Year Plan published by OMB, the Federal Government plans to spend $31 billion on general purpose computers and telecommunications in the FY86 through FY90 period. Without a major initiative with incentives for the development of DAC

mechanisms for a broad range of systems, the existing and future inventory will remain largely vulnerable to attack, at least through the next decade.

c. The survey of DoD computers also indicates that about one-third should be replaced with machines that provide mandatory access control (MAC). The only general purpose computer commercially available today with such capabilities is the Honeywell MULTICS computer. Although we expect to certify MULTICS at the B2 level, the product is not yet on the EPL. Industry has been reluctant to accept the risk of developing computers with MAC capabilities. About five years usually elapses between the time a vendor decides to develop a major new product to the times that the product is commercially availabile. Changes in the security requirements and criteria for certification during the development period can be very expensive. Industry representatives often express concern about their ability to interpret the Criteria in a particular situation. For these practical reasons, many vendors tend to let others pioneer the way. The government should make a major commitment to reduce these risks in order to stimulate development of a significant number of MAC machines during the next 5 years.

d. The Secure Communication Processor (SCOMP), developed by Honeywell Inc., is the only Al-level entry on the EPL. The SCOMP has only limited applications and does not have the processing speed needed to handle the general purpose problem. The national technology base for Al-level systems is essentially non-existent. There do not appear to be even 20 people in the world that have undertaken the essential steps of building an Al system (e.g., a security policy; a security model; a descriptive top level specification; a formal top level specification (FTLS); a detailed design; a formal verification that the design complies with the FTLS, implementation, and test). The other critical Al technology is configuration management. Existing methods are essentially human-intensive, paper-driven approaches that are subject to many classical failures. A great deal of research needs to be done to develop and distribute critical Al technologies to industry before there will be significant numbers of verified systems on the market.

e. The President's 17 September 1984 National Security Decision Directive[2], places great emphasis on reducing the vulnerabilities for Automated Information Systems (AIS). These systems are defined as any system that creates, processes, exchanges, and modifies information in electronic form and includes mainframes, minis, personal computers (PCs), workstations, office automation, data base management systems, local and long-haul network components, distributed operating systems, file server/receivers, and multi media (text, graphics, voice, video) processors. Although we have the Criteria for secure general purpose computers and believe them to be sufficiently general to apply to the other areas, we are only beginning to examine the extent of their applicability to the other segments of AIS's. Because these many components are frequently connected and because security is only as good as the weakest link, a great deal of work needs to be done to understand how to certify systems or to build systems with certified components. Although we can propose ideas, technical feasibility demonstrations are needed. A great deal of research will be required before standards with clear interpretations will be available to provide consistent AIS security across the range of products that comprise a modern information system.

f. To expand on the need for significant research, consider two examples. The DoDCSEC was able to write and publish the DoD Trusted Computer Security Evaluation Criteria between 1981 and 1983, because we had a technical foundation consisting of a decade of research sponsored by the services. This foundation included numerous worked examples to prove the feasibility of concepts and experienced people. We do not have that technology foundation in other AIS security areas. For example:

1. The DoDCSEC is the System Program Office (SPO) for the new multilevel secure, host-to-host encryption device called BLACKER. BLACKER will provide one new technology basis for replacement of the DDN and I-S/A AMPE AUTODIN with multilevel secure systems. The BLACKER program to build host front-ends, Key Distribution Centers, and Access Control Centers, will involve significant RDT&E dollars through the preproduction model. The effort provides extremely important pioneering network security work. Many questions remain to be answered, however, before we can extend the capabilities to include end-to-end encryption and provide support for data transfer rates of future networks as well as provide secure digital, voice, graphics, text, fax, and video multimedia communications being requested today for the command and control of military forces in the future.

2. The ANSI and ISO committees for the Graphical Kernel System (GKS) standard and the Common Language for the Interchange and Processing of Text (CLIPT) standard have not yet considered sensitivity labels that will be needed for multilevel secure, device-independent graphics and text processing. We do not have an industry standard for the internal representation of sensitivity labels used in network devices such as BLACKER. We need research to understand these issues before we publish standards.

3. The Air Force Studies Board sponsored a summer study on multilevel data base security in 1982. Many issues needing work were identified at that time. Essential by no funding has yet been approved to work on these security issues, even though many major system acquisitions need secure distributed data management

systems.

4. Without addressing fundamental issues of trusted computer systems, there are a number of ongoing efforts in the name of security that will give the uninformed a false sense of security. For example, we see complete instructions on hacker bulletin boards for defeating many different dial-back access control implementations.

5. There is a rush to add encryption to workstations and terminals. The fact, however, that information is stored and communicated in encrypted form does not eliminate the computer security vulnerabilities. Often, the perceived benefits of encryption can be circumvented by experts exploiting computer security flaws. Because a TEMPEST tested, encrypted PC costs several thousand dollars more than the comparable "unsecure" system, the Department could invest huge amounts of money for incomplete solutions that will not provide the protection sought and will have to be replaced when more of the community understands Trusted Computer Base (TCB) security issues.

6. Many of the workstations and terminals being considered as candidates for add-on encryption use single-state processors. We know of no way to secure a single-state processor machine. The Motorola 68000 and INTEL 80286 microprocessors have two or more states; however, multilevel secure operating systems are not available for these processors. Several efforts should be started immediately in this area.

The vulnerabilities suggested by these examples also apply to almost every component area of modern AIS. A great deal of research is needed to be able to guide the industry to development of trusted systems.

g. The convergence of telecommunication and computer technolgies encourages rapid aggregation of components to provide interconnected capabilities for sharing information. The unpredicted and explosive growth in the PC market between 1981 and 1984 resulted in sales of over 9 million PC's at a cost of $40 to $60 B. Local area networks are predicted to grow at an estimated rate of 46 percent compounded annually for the next few years. These market trends point to much greater interconnectivity and information accessibility that combine to make information systems more vulnerable today than they were four years ago when the DoDCSEC was founded. Without a significant surge effort, information systems will continue to become increasingly vulnerable to unauthorized access, integrity problems, and denial of critical services.

h. For years, the Department has advocated interoperable command and control systems. Without information sharing, decision making processes crumble and large organizations have difficulty behaving as a single unit. The existence of modern AIS's has been a factor contributing to 147 U.S. industry mergers in 1984 and the emergence of 300 multibillion dollar U.S. corporations in 1985. Today's mergers tend to retain the character of individual profit centers because the heterogeneous characteristics of today's AIS's do not permit horizontal integration outside a single vendor's product line. But horizontal information integration is just the kind of interoperability the DoD needs for the services, components, and U.S. allies to operate synergistically as a single unit. In this regard, the DoD is leading the world in interoperability issues, because, as a $300 billion-dollar-annually corporation, we need horizontal information integration now. Once systems are integrated and integratable, security and integrity will become absolutely essential to corporate survival. A corporation must be able to control data reading, control data writing, and prevent denial of information service. However, the technology base for this class of secure, machine-independent interoperability does not exist today. We have the basic technology concepts, but we need critical proof-of-concept research to demonstrate fundamental protection mechanisms that will prevent unauthorized use, prevent malicious and accidental data change, and complete denial of information services. Generic research in these fundamental areas is needed and the results should be widely shared with industry.

THE CHALLENGE

Based on arguments presented above, we conclude that: (1) the Department cannot afford computer security on a case-by-case basis; (2) computer security requires a fundamental change in the way industry designs and builds computers; and (3) the Department must cause industry to include security as an inherent quality of standard commercial offerings.

The AIS industry is a major national growth industry. AT&T estimates the 1984 gross sales at $141B. IBM estimated the market to be $230-240B annually. Business week estimated 1984 gross sale at $269, growing at 20% annually.

To evaluate the sufficiency of an RDT&E program, we need to understand how the program will change such a huge national growth industry, if at all. In arriving at a decision as to what is sufficient, we need, in addition to global strategy, some metrics to judge the impact of our proposed program. Although not precise metrics, the following factors are important considerations underpinnings for the CSP:

a) Standards are absolutely essential to influence the directions of industry. We have an urgent need for many standards dealing with the various components and aspects of AIS security.

b) Development of consistent standards requires strong central oversight. The National Computer Security Center, under the Director, National Security Agency, has been charged to provide that oversight. Our first priority must be to build a strong center capable of undertaking these new responsibilities.

c) Standards must be consistently applied across the Department and government. If every program office tailors the standard to its unique mission needs, the Department will speak with a confusing voice, there will be no standard, and industry will adopt a "wait and see" attitude. The National Telecommunication and Information System Security Committee (NTISSC), in conjunction with the National Manager and the National Computer Security Center, must develop and enforce standards.

d) A clear capability to monitor compliance with established standards must exist. That monitoring should be fairly and openly applied, especially when we depend on private sector funding for product development. The evaluating organization must be staffed in sufficient quality and numbers to provide responsive and open interaction with industry.

e) The average industry time to bring a new AIS product into the market is 5 years. Standards must remain stable during development periods of this length or industry will not respond by investing private sector dollars.

f) There is little worse than a well-enforced, bad standard. The standards proposed by the National Computer Security Center must be supported by worked examples that prove feasibility, clarify interpretations, and communicate the knowledge and experience to industry. A solid research program, including exploratory and exemplory development is essential.

g) The program must have reasonable balance between near-term, mid-term and long-term objectives. Some issues will take considerable time to resolve. We must not sacrifice the future for near-term fixes, and we need to do the best we can to protect the current and planned inventory through its remaining useful life.

h) The job facing the government is that of building and distributing to industry a fundamental new technology. The job must be completed before the underlying assumptions are obsoleted by changing technology. The program must have sufficient industry participation to have a significant impact on future directions and technology decisions by industry and the private sector.

PRIORITIES

Our first priority under the President's Directive will be to build a strong National Computer Security Center. We will need a strong in-house capability to provide the technology basis for standards development, to support a significant product evaluation capability, and to foster much wider awareness of computer security needs and issues. The in-house RDT&E capability will permit the best possible progress with limited resources and in the event additional monies become available provide the technical oversight for a greatly increased industry participation. Planned personnel increases should enable the Center to support approximately 45 new commercial product evaluations by 1990. This would be a significant increase over 1985 levels and should send a strong signal to industry that will encourage investment of private sector dollars in trusted product development.

Our second priority will be to greatly strengthen the research effort by strengthening industry participation. Our strategy includes three thrusts: 1) near-term efforts to improve security of the current inventory of government computers; 2) mid-term efforts to greatly increase the availability of trusted products with much better security features than are generally used today; and, 3) long-term efforts to develop and distribute to industry the technology base needed to build much more trustworthy system than we currently know how to build.

In carrying out this strategy, we are concerned that near-term fixes do not jeopardize long-term solutions. To achieve this objective, a panel that was convened in response to a Secretary of Defense security initiative recommended that the five-year computer security research program be allocated as follows: 20 percent of the resources applied to development of C-level DAC add-on security capabilities to protect current and planned systems, 30 percent to stimulate development of B- and A1-level MAC systems and 50 percent to extend our understanding of assurances beyond A1. The Technical Review Group recommended a 30 percent, 30 percent, and 40 percent mi, repectively for these objectives in the transition year FY87. The emphasis on near-term dollars will be to develop working products that will improve security for immediate needs and encourage private sector development and marketing of similar or better products. The mid-term efort will focus on exemplary products in the public domain to greatly reduce the risk involved in interpreting standards. These exemplary multilevel secure implementations will be made widely available to industry to accelerate the availability of new products and stimulate private sector development of better products. The government expects to carry the initial research burden to extend our knowledge of beyond A1. Given the fact that formal software and hardware verification may not mature to affordable technologies for many years; given also the facts that configuration control may be our only alternative to reduce vulnerability of DAC mechanisms, will be required in any case and to extend assurances beyond A-1; and,

given the fact that configuration control is emerging as a fundamental need of network security not adequately supported by current methods; we hope to significally increased our efforts in formal methods and automated configuration control.

Beginning in Fiscal Year 1987, a major RDT&E initiative is being planned to develop a new technology base in computer security and to distribute that base to industry. The effort will be carried on with broadened industry participation. Our future program will comprehensively treat all aspects of AIS security from the component and the total systems view.

CONCLUSION

The Center's program provides a sound basis for expectations that computer security vulnerabilities could be greatly reduced by the end of the 15-year period. It also provides a balanced effort to reduce vulnerabilities in the intervening years. The research is conducted on a schedule that would significantly contribute to the long-term interoperability goals of the DoD.

REFERENCES

1. DoD Computer Security Center, Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83, dtd 15 August 1983.

2. The White House, National Policy on Telecommunications and Automated Information System Security, National Security Decision Directive 145 (Unclassified Version), dtd 17 September 1984.

NATIONAL BUREAU OF STANDARDS' COMPUTER SECURITY, INTEGRITY, AND RISK MANAGEMENT PROGRAM

Dr. Dennis K. Branstad
NBS Fellow

and

Dr. Stuart W. Katzke
Computer Scientist

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

Computer security is a critical component of the overall management of computers. The National Bureau of Standards (NBS) through its Institute for Computer Sciences and Technology (ICST) initiated a Computer Security and Risk Management program in 1972. Since that time, numerous standards, guidelines, and technical reports have been issued in the areas of physical security, technical security, and computer security management. The program encompasses research and development of security standards, transfer of technology to potential implementors and vendors, and assistance to users of security technology.

NBS/ICST interacts with Federal agencies, voluntary standards making organizations, and private industry in this program. Federal organizations utilize the technical standards and expertise established within the program. Federal executives have requested assistance in addressing computer security problems. Congressional organizations, including GAO, OTA, CRS and several congressional committees, have requested publications, briefings and testimony regarding computer security. Finally, ADP vendors, security product vendors and security consultants interact with, and use the results of this program.

Computer security events, ranging from computer "hackers" to international terrorism, have raised the awareness of computer vulnerabilities and risks. Significant risks have been identified in defense/intelligence systems, Electronic Funds Transfer systems, automated decision making systems, and real-time control (e.g., air traffic control) systems. ICST has structured a comprehensive program in computer integrity (detecting unauthorized entry or change of information), confidentiality (preventing unauthorized disclosure of information) and reliability (assuring availability of information processing) to reduce existing vulnerabilities and risks. Organizations now realize that they have become totally reliant on computers; in many cases, it is impossible to return to manual methods when the computers are unavailable, unreliable or unsecure.

ICST draws upon its own research and that of other organizations in accomplishing its goals. Generically, technology transfer interfaces have been established linking vendors and users, government and industry, managers and technologists. Specifically, the following projects depict the depth and breadth of this program:

INFORMATION SYSTEMS SECURITY LABORATORY. The Information Systems Security Laboratory at the NBS was established to provide a research environment in which various computer security technologies and techniques can be studied and tested. The Laboratory is intended to support activities in areas such as communications network security, cryptography, personal identification, access controls, secure system architectures, and related computer security disciplines.

The objective of the Laboratory is to enable NBS to maintain currency in the rapidly changing technologies that will impact security of computers and information in the years ahead. Among the current activities of the Laboratory are the following:

- Personal Computer Guidance - NBS Special Publication 500-120, "Security of Personal Computer Systems: A Management Guide," was the first in a series of documents intended to provide an understanding of the information security threats involved in using personal computers and of approaches to reducing the associated risks. A user guide, technical guide and FIPS Guideline are planned for the near future.
- Security Products Data Base - A comprehensive database of security-enhancing technologies, products, and services is being built. This is being done in support of efforts to develop a FIPS Guideline on the security of small computer systems.
- Personal Computer Security Product Testing - With the co-operation of industry, a wide range of commercially available computer security devices

designed for personal computers, communications security, and related purposes is being examined. This has enabled NBS to develop better guidance for users of computer systems who do not have the resources to develop and build their own security mechanisms.

RISK MANAGEMENT. Research is underway to develop an automated, expert system that will allow iterative safeguard selection based on the cost of controls and their relative reduction of risk. Current activities are focusing on the development of a conceptual model of the risk management process which can be used to foster a common understanding of the terminology and to describe the functional relationships that exist between key elements such as assets, vulnerabilities, threats, safeguards, threat frequency and severity, and impacts.

CONTINGENCY PLANNING. Previous NBS contingency planning publications consist of a Federal Information Processing Standard (FIPS) Guideline and an Executive Guide. Work in progress includes the development of a guide for meeting the backup requirements of computer applications and systems. The guide identifies requirements criteria that must be addressed when considering backup alternatives for computer systems and uses the criteria as a framework for describing the advantages and disadvantages of alternative backup choices.

CERTIFICATION AND ACCREDITATION. NBS is participating in a joint project with the President's Council on Integrity and Efficiency aimed at developing criteria for prioritizing system development life cycle audits. This work has been strongly supported by both Federal and private organizations. Publications resulting from this work will include a workshop report on auditor work priority criteria, a report on an auditor work priority scheme, and a guide to auditing security and controls throughout the automated information system life cycle.

LOCAL AREA NETWORK SECURITY. An experimental secure Local Area Network (LAN) has been established to investigate the methods of protecting the security and integrity of information processed in this type of environment. The project is being conducted in conjunction with the telecommunications security organization of the Department of Defense. A secure transport layer protocol is being implemented to protect information between computers in the network.

DATA ENCRYPTION STANDARD. NBS published the first Federally approved public standard for the cryptographic protection of computer data in 1977. This standard was needed as a basis of protecting communications of information in a general public data network. The standard has gained wide acceptance and usage. More than ten additional Federal, Industrial and Commercial standards have been issued that utilize the Data Encryption Standard (DES).

ELECTRONIC FUNDS TRANSFER SECURITY. A number of security standards have been developed within the Financial Community using the DES. NBS has provided a technical leadership role in developing, implementing, and validating these standards. ANSI X9.8, X9.9, and X9.17 all utilize security technology that was provided, in part, by NBS. These standards are for protecting Personal Identification Numbers, protecting Electronic Funds Transfer Messages and providing automatic cryptographic key distribution, respectively.

PASSWORD USAGE STANDARD. A standard on the use of passwords for personal identification and authentication was recently approved for publication as FIPS 112. The standard specified ten factors that must be considered when designing and implementing a password system. It also defines minimum security criteria for each of the ten factors that must be met in Federal applications.

SECURE "SMART CARD" TECHNOLOGIES. NBS recently held a workshop on the security aspects and requirements of Integrated Circuits (ICs) on credit-cards. These cards are assigned to an individual and carried by the individual. Many applications of the cards were discussed with respect to their security requirements and benefits. Research and standards activities were outlined for secure chip cards.

NETWORK SECURITY ARCHITECTURES. NBS has participated in identifying the requisite areas of network architectures for implementing a variety of security provisions. A workshop was sponsored by the DoD Computer Security Center last Spring in computer network security. The Open Systems Interconnection network architecture is the model commonly used for this work. An addendum to the International Standards Organization document on the architecture is being prepared regarding security.

DOE COMPUTER SECURITY CENTER ACTIVITIES

Lara H. Baker
Los Alamos National Laboratory
Los Alamos NM 87545

ABSTRACT

This paper is a brief summary of a panel discussion at the 8th National Computer Security Conference. Panel members included Dr. Lara H. Baker, Project Manager for Information Security, Los Alamos National Laboratory, Los Alamos, NM; Mr. Charles M. Cole, Deputy Associate Director for Computer Security, Lawrence Livermore National Laboratory, Livermore (LLNL), CA; and Mr. Duane G. Harder, Project Leader for Center Activities, Los Alamos National Laboratory, Los Alamos, NM (currently on detail to Washington DC). Each panel member discussed a separate subject: Dr. Baker mentioned the background, mission, and composition of the DOE Center for Computer Security; Mr. Cole discussed computer security activities at the Lawrence Livermore National Laboratory; and Mr. Harder discussed recent activities in the DOE Center for Computer Security.

BACKGROUND

The U.S. Department of Energy (DOE) has a long-standing interest in computing and computer security. Indeed, DOE, and its predecessor agencies, were interested in computer security long before much of the community realized there was such an animal. To a first approximation, the DOE has in its inventory, and in the inventory of its contractors, about 25 percent of the computing systems owned or operated by the federal government. Furthermore, because of the nature of its activities, DOE has substantially more than 25 percent, perhaps as much as 50 percent, of the computational capability within the United States government. For example, Mr. Cole discussed the LLNL's computing suite, which includes 6 CRAYs, 4 CDC 7600's, several score of DEC VAXs, and several hundred DEC PDP-11's. The suite at the Los Alamos National Laboratory is substantially identical. And these Laboratories are not, by any means, the only computing resources in the Department. Thus, because of DOE's long-term interest in computing, and the fact that some of the information in those computers is among the most sensitive in the nation's inventory, DOE has had a long-standing interest in protecting that information.

In 1980 the DOE Office of Safeguards and Security formalized its computer security R&D and field assistance efforts by forming the Department of Energy Center for Computer Security. For various historical reasons, this Center was placed at the Los Alamos National Laboratory. The mission of the DOE Center for Computer Security is to maintain a center of excellence in computing security for the Department, to provide Research and Development (R&D) activities as appropriate for upcoming Department missions, and to provide assistance to DOE and DOE contractors on an as-needed basis. Since 1980, the Center has substantially increased its assistance role to the Department and to DOE contractors, while maintaining an R&D role.

Because the DOE has chosen to run very large activities with a minimal headquarters staff, several organizations, including the Los Alamos National Laboratory and Lawrence Livermore National Laboratory are captive contractors of the Department of Energy. As a result, each year DOE/OSS and Los Alamos agree on the tasking of the Center for the next year, DOE provides the funding, and Los Alamos manages the Center's activities.

COMPOSITION

The DOE Center for Computer Security is a rather close-knit cadre of about 10 people, or more precisely 10 full-time equivalents, located at Los Alamos, and occasionally actually found in Los Alamos and not on travel. However, the DOE's computer security effort is substantially more than the DOE Center for Computer Security. As in other Departments, the people in the field, who are actually running the systems, form an integral part of the computer security effort. There are 10 Computer Security Operations Managers (CSOMs) who have overall responsibity for computer security in the DOE field offices, and for the contractors who report to those field offices. In addition, each classified computing system, or set of systems at one location, in the Department of Energy, or at DOE contractors, has a Computer System Security Officer (CSSO) assigned to be responsible for the security of that system. The CSSO is the cornerstone of the DOE computer security effort.

RECENT ACTIVITIES OF THE DOE CENTER FOR COMPUTER SECURITY

Since its inception, the DOE Center for Computer Security has had a considerable effort involved with advice and assistance to DOE and DOE contractors through a newsletter, and through R&D activities. These efforts continue, perhaps with some emphases changing over time.
The Center's newsletter is published quarterly and currently has a mailing list of about 1000 names. It is unclassified and provides a forum for discussions among the computer security practitioners, the DOE and DOE contractors.

The Center also provides instructors for classes designed to assist CSSOs in discharging their obligations. These classes are held about quarterly, and usually consist of 30 to 60 people. The Center provides two instructors; DOE Headquarters also provides one or two instructors. The classes are three days long and are held at various sites around the country. Despite the fact that the schedule calls for these classes to be held only quarterly, the demand is quite high, and we have just completed the fifth such class this fiscal year.

As part of the DOE's operations security program, teams from the DOE Center for Operations Security at Los Alamos do vulnerability assessments (VAs) on various DOE sites. These VAs involve taking an adversary view of the site and looking into what information is available to a competent, trained, but uncleared, outsider. Computer security is an integral part of this assessment, and staff members from the DOE Center for Computer Security often participate as technical experts in VAs. These VA visits provide an interesting insight into what is going on in facilities and into the needs of the various facilities. Many things observed on VAs, and on other visits, have resulted in changes in CSSO classes and in other training.

As part of its overall responsibilities, the DOE is involved in activities involving about 10 billion dollars in facilities scattered over all the United States. Interconnecting these facilities with high-speed data transmission is an ongoing effort in DOE. The initial stage of this interconnection is called the wide-band communications network (WBCN). It will consist of 56-kilobit lines connecting major DOE sites and will be used principally for CAD/CAM work among DOE facilities. The DOE Center for Computer Security is responsible for the development of a security controller to allow access and authentication among the DOE sites over the WBCN. This is an interesting exercise in that, while all the DOE sites are, in fact, under one set of security guidelines, the WBCN is actually interconnecting a large number of sites that have different security policies. Assuring the security of this collection, a network aggregate, is a very interesting problem.

COMPUTER SECURITY AT THE LAWRENCE LIVERMORE NATIONAL LABORATORY

The Lawrence Livermore National Laboratory (LLNL) Computing Center (LCC) comprises one of the worlds largest concentrations of computing power. This computing power is used in various programs at LLNL, including energy development, pure research, SDI, and weapons development. The aim of the computer security program at LLNL is to provide the greatest possible access to approved users, and the least possible access to anyone else. This presentation includes an overview of what is in that center and how it is protected.

ON THE INTEGRITY PROBLEM

Sig Porter
Terry S. Arnold

Merdan Group

4617 Ruffner St.
San Diego, CA 92117

Abstract

The term 'integrity' has been used to express different concepts without explicitly noting the differences. Explicitness about the nature of integrity clarifies the design process. Six types of integrity are discussed here. Two types are of particular value for system design and specification.

Introduction

In the years following the appearance of Biba [1] a great deal of discussion has occurred in the area of what is meant by integrity. At times it has appeared that each member of the computer security community has their own (usually implicit) definition, and that these definitions are different. The primary goals of this paper are to present an explicit set of definitions, and to introduce the concept that there are several kinds of integrity that must be thought about. The secondary goal is to present a base for reasoning about the different kinds of integrity.

This paper is limited to security concerns in the automated part of systems. We do not address considerations of design validation, or interpretion of the intentions of message senders. We also do not consider clearances or integrity of people.

Definitions

One reason for the confusion about integrity is that it is not a simple one-dimensional variable. Some candidates for integrity dimensions are given below. These will then be reduced to a smaller number.

1. How correct (we believe) the information in an object is.

2. How confident we are that the information in the object is actually from the alleged source, and that it has not been altered from the original form.

3. How correct (we believe) the functioning of a process is.

4. How confident we are that the functioning of a process is as it was designed to be.

5. How concerned we are that the information in an object not be altered.

6. How correct we hope the information in an object is.

The environment we used for this study includes message transmission over an unreliable/hostile medium, in addition to the usual computer object considerations. We believe the extreme conditions of this environment help to illuminate the situation.

Discussion of Definitions

Dictionaries list three definitions for integrity:

1. Completeness, wholeness
2. Unimpaired condition, soundness
3. Honesty, sincerity

Definition 3 applies to people, and is beyond the scope of this paper. The other definitions appear to be the source of the binary view of integrity ("An object either has integrity or it doesn't"). Integrity-1 seems to have resulted from an attempt to generalize this binary view of integrity. While there are several possible ways to make the notion of correctness concrete, e.g., correct design, the sender sent what he meant to send, N bits of an M-bit message were correct, etc., we find it most useful to consider the ways in which the correctness of a message can be validated: one can check the syntactic form or interfield consistency of certain rigidly formatted messages.

Information theory limits the usefulness of correctness as an integrity measure, since conveyance of information and complete verification of correctness are mutually exclusive.

Correctness is more meaningful for functions (i.e., hardware and software) than for data. The difference is that for functions our concern is for correctness of design, rather than communication of information. By definition, Trojan horses are not part of a design, since the purpose of the design is to accomplish the aims of the system's user or customer. We regard Trojan horses as an implementation failure, instead. Integrity-3, then, comprises correctness of design, correctness of implementation, and unalteredness of the implementation. Integrity-3 validation includes structured walk-throughs and formal verification. Further consideration of the integrity of design, while necessary, is beyond the scope of this paper.

Integrity-2 is based on standard Bayesian probability calculations. Integrity-2 validation mechanisms include error detecting and correcting codes, cryptographic authenticators, and digital signatures.

Cryptographic authenicators have, themselves, been a source of confusion, since some workers have ignored the distinction between an object and a second object created by encapsulating the original object (say, by adding a validation field). We will call this second object

a capsule. Integrity-6 resulted from our examination of an attempt at a formal specification which appeared to apply a high integrity level to a capsule immediately upon receipt from an unreliable source. This high level seemed to be based on expectations about the (to be determined) integrity-2 of the object in the capsule.

To put it more formally, the capsule is received with high integrity-6 (because we are prepared for an important and meaningful message) and low integrity-2 (because the capsule was received from an unreliable source). If the validation process is satisfied, then the content (of the capsule) has high integrity-2, and integrity-6 is not meaningful. If the validation process is not satisfied, then the content has low integrity-2, and integrity-6 is not meaningful. Since integrity-2 is meaningful both before and after validation, it is greatly preferable to integrity-6. If we do not distinguish between integrity types, and assign high integrity values to both capsule and content, then we will have made a hidden semantic shift (from 6 to 2) which will interfere with relating formal specification to the real world. If we use integrity-2, then validation is an integrity raising process, which is automatically flagable (as it well should be) as a trusted process.

Since integrity-6 may look like a straw horse, we note that we didn't invent it (except possibly by misinterpretation of the work of others).

Integrity-5 applies to electronic funds transfer, and to military command and control. The use of integrity-5 can be illustrated by the message example: The incoming capsule has moderate integrity-5. It's not extremely high because if the capsule has been altered we will detect it at validation. It's not low because we will be inconvenienced if the capsule has been altered. That is, we may be unable to extract the capsule content and thus require further countermeasures, such as retransmission. The validated message content has high integrity-5 since alteration after validation may not be detected. We also observe from this example that integrity and denial of service are not orthogonal. Thus an intended attack on integrity becomes an actual denial attack.

Integrity-4 is concerned with delivered and running software, but not with the design process. If the data representing the code of a function or process has not been tampered with, then its behavior will be as designed and implemented. Validation mechanisms for integrity-4 include configuration management and program data protection. This data protection is identical with the concerns of integrity-2. Thus, except for implementation considerations, integrity-4, is identical with integrity-2.

Since we would like to minimize the number of types of integrity with which we must be concerned, we first eliminate integrity-3 and 4, until we have a way to deal with design issues. We regard 6 as dangerous, and therefore eliminate it. Integrity-1 is occasionally relevant, but its utility is severely limited by information theoretic considerations. The intent behind integrity-1 can usually be better handled with integrity-2. Integrity-5 is quite different, and quite valuable.

We find ourselves using integrity-2 and integrity-5 only, because integrity-2 (confidence) is more manipulatible than integrity-1 (correctness). Integrity-2 can, in fact, be measured as probability, using all the customary probability calculations. For both integrity-2 and integrity-5 validation is an integrity raising process.

Integrity-2 is appropriate to use for system design goals. Integrity-5 may enter as part of the design process, to help in determining countermeasures. At system run time, integrity-2 is an appropriate measure to apply to actual objects.

Designing for Integrity

When designing the integrity aspects of a system, we first consider how much we care if a data item (or message) of a particular class is altered. This is integrity-5. To say we care a lot means the same as attributing a high integrity-5 to the item.

Integrity-5 is a non-quantifiable value with an impact (and relation with integrity-2) which depends on the environment. For example, consider an intermediate numeric value of an internal (to cpu) calculation: High integrity-5 may mean that we have an integrity-2 goal of .99999999 probability that the value has not been altered, but no special measures are required. Conversely, for a message which has been received from a noisy and hostile transmission path, we may set an integrity-2 goal of .99 probability that the value has not been altered, and require both error correction codes and authentication fields to achieve this.

If our task is generation of specifications, we will probably set integrity-2 goals, based on integrity-5. If our task is design based on integrity-5 specifications, then our first analysis may be to compute integrity-2 assuming no (or nominal) countermeasures. We would then (subjectively) compare this integrity-2 value with the integrity-5 specification, and determine if further countermeasures are appropriate.

There are several possible approaches to making integrity-5 more formal:

1. Relate integrity-5 levels to countermeasure requirements (as in orange book). [Considering the above remarks about the impact of integrity-5 depending on the environment, this is probably not a good idea.]

2. Relate integrity-5 levels to specific integrity-2 values. [Also not too good, since these are different concepts.]

3. Establish classes of environments (benign, hostile, open closed, noisy, etc.) and set up a matrix with coordinates of environment classes and integrity-5 levels.

Fill the intersections with countermeasures or integrity-2 values (as in 1 or 2, above.) The additional coordinate may make this reasonable.

Reasoning About Integrity

Since we have established definitions of the kinds of integrity, we can now discuss the kinds of reasoning that are appropriate for verification of implementations. Under the assumption that levels of integrity-2 were included as part of the specifications, we can calculate the actual performance of each of the countermeasures employed. I.e., for each countermeasure we can calculate the probability that it could be defeated in an undetectable or nonrecoverable manner. These calculations would be based on the same assumed channel model used in the definition of the integrity-2 for the specifications. The achieved integrity-2 levels (expressed as numeric probabilities) are compared arithmetically to the specification levels. An obvious decision rule is that for a given countermeasure the achieved probability (of undetected modification) must be less than the specified level. It should be noted that we are talking about the same things at both the specification and implementation verification levels of abstraction. Introduction of a partial ordering of different integrity-2 levels does not appear to be of value and obscures the issue. This appears to be due to the fact that even though integrity-2 levels can be ordered (since they are numbers) the entities to which they relate may not be comparable in any reasonable sense.

Conclusions

There are two useful and significantly different types of integrity, and they are

2. How confident we are that the information in the object is actually from the alleged source, and that it has not been altered from the original form.

5. How concerned we are that the information in an object not be altered.

We have been using these concepts for a number of months on some real applications and are finding them very useful. We find that the knowledge of these integrity types helps to avoid inadvertent changes in definition in the middle of a specification. As a result, we find it much easier to keep our design concepts clear and directly convertible to formal expression.

Reference

[1] Biba, K. J., Integrity Considerations for Secure Computer Systems (Technical rept.), Mitre Corp, Bedford Mass, Report No.: MTR-3153-REV-1; ESD-TR-76-372, Apr 77, 66p

A PRACTICAL ALTERNATIVE TO HIERARCHICAL INTEGRITY POLICIES

W.E. Boebert

Honeywell Secure Computing Technology Center
Minneapolis MN

R.Y. Kain

University of Minnesota
(Consultant to Honeywell)
Minneapolis MN

BACKGROUND

The Secure Ada Target

The Secure Ada Target (SAT) project is an effort to develop a machine which meets and exceeds the A1 level of the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC). An overview of the machine is given in Reference 1.

Enhanced Security Policies

The SAT system design meets the A1 requirements with respect to the mandatory and discretionary policy requirements, and it exceeds the A1 level by enforcing an enhanced mandatory policy whose aim it is to prevent corruption of sensitive information. Early versions of the machine incorporated a variant of the "traditional" hierarchical integrity policy; detailed analysis showed the inadequacy of this approach, and an alternative based on types and domains was developed.

PROBLEM STATEMENT

TCSEC Requirements

The TCSEC requires that systems at the B2 level of assurance and above demonstrate conformance to a security policy. The TCSEC further gives a set of minimum requirements that an acceptable policy must meet. Briefly stated, these requirements are that information be labelled internally with a security level, and accesses made by active subjects to information-holding objects be restricted in a manner that prevents information from flowing down in security level. We shall refer to this policy as the "compromise policy," and the security level used in its policy decisions as the "compromise level" of objects and subjects.

The TCSEC is silent on the equally important topic of preventing the corruption of sensitive information. A modular implementation of the TCSEC requirements dictates that it is necessary to impose proven constraints on information flow other than those imposed by the mandatory policy. This implication arises because the TCSEC requires that exported information be properly labelled with its compromise level. A modular implementation of this exportation process would have separate modules for label insertion and device control.

Practical secure systems also require constraints on information flow in order to defend against so-called "virus" attacks, to demonstrate assured data flow through cryptographic devices, and to enforce sophisticated security policies whose aim it is to prevent aggregation and inference.

First Efforts

An early response to the problems of information corruption was the development of "Integrity Policies," several variations of which are described in Reference 2. In effect, these policies add a second attribute to information (integrity level) and impose

access restrictions in order to protect sensitive information from unauthorized modification.

INTEGRITY POLICIES

Varying Integrity Levels

The policies described in Reference 2 fall into two broad classes. In the first class, the integrity levels associated with subjects and objects may change. This class includes the Low-Water Mark Policy for Subjects and the Low-Water Mark Policy for Objects.

In the Low-Water Mark Policy for Subjects, a subject may neither modify objects nor send messages to a subject whose integrity level is greater than the one the sender currently has. The current integrity level of a subject is equal to the lowest integrity level of any object to which it has been granted observe access; hence the name "Low-Water Mark." "Execute" access is treated as a form of observe.

The Low-Water Mark Policy for Objects does not impose any restrictions on the ability of subjects to modify objects. Instead, the current integrity level of an object is set to the lowest integrity level of any subject which has been granted "modify" access to that object.

Integrity policies in the above class have seen little, if any, practical use, owing to the difficulties of administrating them and the pathological states which they allow (such as a subject being denied access to objects it has created.)

Fixed Integrity Levels

The second broad class of integrity policies includes the Ring Policy and the Strict Integrity Policy. In these policies, the integrity levels of both subjects and objects are fixed. Under the Ring Policy, a subject may obtain "observe" access to any object, but may not modify objects nor communicate with subjects of higher integrity. The Strict Integrity Policy is the full formal dual of the compromise policy defined in the TCSEC. It consists of a Simple Integrity Condition, which states that a subject cannot observe objects of lesser integrity; an Integrity *-property, which states that a subject cannot modify objects of higher integrity; and an Invocation Property, which states that a subject may only send messages to subjects of higher integrity.

This second class of integrity policies has fewer intrinsic difficulties than the first, and variants have been implemented in reference monitors.

General Principles

Both classes of integrity policies represent varying interpretations of the same general principle: information should only flow "up" in integrity. In order to avoid excessive detail, we will offer our critique of, and alternative to, the general class of policies which adhere to this principle. We will call such policies "hierarchical integrity policies." This class includes all policies which assign an attribute called "integrity level" to information, and which then impose rules to prevent (to one degree of assurance or another) information at high integrity levels from being corrupted by information of low integrity.

Integrity and Compromise

It is tempting to view hierarchical integrity policies as duals or complements of the compromise policy mandated by the TCSEC. While such a relationship can be shown to exist formally (especially in the case of the Strict Integrity Policy), the relationship does not exist in the broader sense of intent and application.

In particular, the nature of a compromise policy is that controls are imposed on programs based upon the context in which they execute, and not upon the degree of trust placed in the programs themselves. In

particular, a compromise policy such as that mandated by the TCSEC can be shown to prevent the compromise of information even if the programs being executed are hostile in their intent.

Such immunity from hostile programs cannot be obtained by using integrity policies. If there were a hostile program in the system, it could simply wait until it was executing in the context of a high-integrity subject and then work its damage on high-integrity information. Under the Low-Water Mark Policies and the Strict Integrity Policy, this danger is prevented by assigning integrity levels to programs and equating "observe" and "execute" access. In these policies a high-integrity subject is therefore bound to executing high-integrity programs. In the Ring Policy no such restriction exists, and the policy is trivially subvertible by Trojan Horse techniques.

From the above it can be seen that there is an essential difference between compromise and integrity: compromise level is more naturally bound to subjects and integrity level is more naturally bound to programs. Attempts to bind integrity level to subjects, as is done in the above policies, should lead to difficulties in application. We will show that such difficulties do in fact exist; they manifest themselves as an excessive need for the concept called "trust."

Trust

A "trusted subject" is one which is privileged to selectively violate the letter of a particular policy. The programs executed by the subject must be verified to insure that the exception does not violate the intent of the policy. This in turn requires that the intent of the policy be explicitly stated; this is often no easy matter.

In the case of compromise policies, trusted subjects are those which are permitted to "write down," that is, to cause information to flow downward in compromise level. In the case of such subjects, the adherence to the "higher" policy is demonstrated by showing that the subject moves a trivial amount of information, that the movement of information is audited so that abuses can be detected, and/or that the movement takes place at the instigation of an authorized user (a so-called "downgrader").

If we follow the pattern of viewing integrity policies as the formal duals of compromise, then "integrity trust" is the privilege of "writing up" in integrity. As with compromise, we associate trust with "modify" access in order to simplify the discussion.

The attribute of trust, in the policies under discussion, is bound to subjects and not to programs. It is therefore necessary to prove that trust can never be abused; that is, that no hostile program can ever be executed within the context of a trusted subject. This in turn requires verification of usually complex low-level mechanisms which bind programs to subjects.

It is also necessary to state the intent of the policy being enforced, and to formulate a subject-local property which captures that intent. It is then necessary to verify that the property is exhibited by all programs which could be executed in the context of the trusted subject. The use of trust therefore greatly complicates the proof process and reduces the degree of assurance in the system. It is accordingly a goal of the SAT effort to reduce the use of trust as much as possible, and it was this goal that led us to question and finally discard the notion of a hierarchical integrity policy.

CRITIQUE

Assured Pipelines

In this section we will present a critique of hierarchical integrity policies. We will consider the shortcomings of such policies in the context of what we call an "assured pipeline," a subsystem which is security-relevant and which must be encountered by data flowing from a particular

source to a particular destination. Examples of assured pipelines are labellers and cryptographic subsystems. In Reference 3 we give an example of a similar subsystem which does not transform data, but instead selectively audits requests made to the reference monitor.

A labeller is a verified subsystem which converts the security level of an object from internal form to external form prior to the export of that object. The most common instance of a labeller is one which prints the classification level of a single-level object at the top and bottom of the pages when that object is output to a hard-copy device. A cryptographic subsystem encodes data in such a way that it may be safely downgraded and transmitted over an insecure communications path without effectively declassifying the information contained within that data.

From the above discussion, it can be seen that assured pipelines represent the most basic kind of structure which one would wish to construct and prove secure in a Trusted Computing Base.

Security of Assured Pipelines

To prove that an assured pipeline is secure requires the demonstration of three properties:

1. The transforming subsystem cannot be bypassed. That is, no hard-copy can be printed without labels, and no information can go out on the insecure path in unencrypted form.

2. The transforms cannot be undone or modified once done. Data cannot be intercepted between labelling and printing, and have the labels removed; data cannot be intercepted between encryption and transmission, and have unencrypted information inserted.

3. The transforms must be correct. The labeller must insert external labels which are the proper representation of the internal label of the object; the cryptographic subsystem must properly implement the desired cryptographic algorithm.

The last property is the only property amenable to program proof techniques; the first two properties must be demonstrated by recourse to some global attribute of the underlying system. We will now show that enforcement of a hierarchical integrity policy is a poor candidate for such an attribute.

Integrity and Assured Pipelines

For simplicity, we shall use the labeller for hard-copy output in our discussion. Other labellers and cryptographic subsystems pose the same problems for hierarchical integrity policies; only the terminology used in the example will change.

There are two object types and two modules in this example of an assured pipeline. The object types are unlabelled and labelled data; the modules are the labeller and the output subsytem. Unlabelled data does not include the printable classification levels at the top and bottom of pages; labelled data does. The labeller determines the security level of the object from its internal label, locates page boundaries, and inserts the proper label text. The output module is a device driver which causes the labelled data to appear on some appropriate hard-copy device.

The local security properties which must be proven of each of the modules are that the labeller selects the proper printable label and puts it in the proper place, and that the output module moves data to hard copy without modification to the label text.

The global security properties which must be proven of the pipeline are:

1. Only the labeller module produces labelled data.

2. Labelled data cannot be modified.

3. The output module will accept labelled data only.

We will now show that attempts to enforce these properties using a hierarchical integrity policy will inevitably involve the use of "trust" somewhere in the pipeline. Note that all information is at the same compromise level, so that the mandatory security policy imposed by the TCSEC is trivially satisfied.

There are three alternatives to assigning integrity levels in such a pipeline: the integrity levels of all data may be equal, the integrity levels may increase as data moves toward the output device, and the integrity levels may decrease as the data moves down the pipeline.

If labelled and unlabelled data are at the same integrity level, then no integrity policy will be able to distinguish between them. A hostile program will be able to remove or modify labels at will between the labelling and the output steps, and the output module will not be constrained by integrity level to outputting only labelled data.

If labelled data is at a higher integrity level than unlabelled data (the intuitive case), then trust must be invoked at each module in the pipeline, as it is clear that in such an arrangement information is flowing "up" in integrity.

The case where labelled data is at a lower integrity level than unlabelled has the same shortcomings as the equal integrity level case.

Thus the application of hierarchical integrity policies to the most basic structure of a secure system either fails to enforce the desired restrictions or requires an exception from the policy at each step. We argue that this situation represents an excellent definition of the word "impractical," and offer an alternative that avoids these shortcomings and confers other benefits as well.

POLICY ENFORCEMENT IN THE SECURE ADA TARGET

The SAT machine directly implements the reference monitor mandated by the TCSEC. The SAT reference monitor system checks every individual access attempt for consistency with the security policy being enforced by the system.

The SAT reference monitor is implemented in hardware, and resides between the processor, which generates memory access requests, and the memory system, which satisfies these requests. The reference monitor intercepts illegal access attempts; an interrupt is caused when an illegal access is detected. For "normal" checking, the system aborts the offending subject, thereby guaranteeing that no illegal accesses can be completed and further that the program cannot obtain much information regarding the security state of the system by repeated attempts to make illegal accesses. (Otherwise, the system's security state might be used to construct a covert channel between two subjects.)

The SAT reference monitor is implemented by a combination of a memory management unit (MMU), which has conventional rights checking facilities, and a tagged object processor (TOP), a new module responsible for the system's protection state and the enforcement of that state. In particular, the TOP sets up the tables that define the access rights checked by the MMU. For system integrity, it is also necessary that the TOP be responsible for resource management and for the integrity of the internal state of the reference monitor. One important part of this state is the global object table (GOT), which contains a description of the security attributes of all objects within the system. In general, all elements of the system, including users, security properties, code, and data, are objects described within the GOT and managed by the TOP.

Of major concern are the security attributes of objects and their use in determining the access rights to be placed within the MMU during program execution. The basic SAT design starts with a minimum set of security

attributes sufficient to satisfy both the mandatory and discretionary security policy requirements, which require comparisons between attributes of the subject in whose context a program is executing and attributes of the object to be accessed by that program. Thus security attributes are associated with both subjects and objects, and the TOP must make appropriate comparisons to establish proper access rights in the MMU.

Three security attributes are associated with subjects and three different attributes are associated with objects. Both subjects and objects have security (compromise) levels. Each subject is performing its function for some "user," whose identity is the second subject security attribute. The corresponding object attribute is its access control list (acl), which lists those users who are allowed access to the object's contents, along with the maximum access rights that each designated user is permitted. The third subject security attribute is the "domain" of its execution, which is an encoding of the subsystem of which the program is currently a part. The corresponding object security attribute is the "type" of the object, which is an encoding of the format of the information contained within the object.

The process of determining the access rights to be accorded a particular subject for access to a particular object uses all of these three security attributes, as follows.

To enforce the mandatory access policy, the TOP compares security levels of the subject and of the object, and computes an initial set of access rights according to the algorithm defined in Section 4.1.1.4 of the TCSEC.

To enforce the discretionary access policy, the TOP checks the acl for the object; the acl entry that matches the user portion of the subject's context is compared against the initial set of access rights from the mandatory policy computation. Any access right in the initial set which does not appear in the acl is deleted from the set. The result is an intermediate set of access rights.

The third SAT access rights determination check compares the subject's domain against the object's type. Each domain is itself an object, and one of its attributes is a list of the object types accessible from the domain and the maximum access rights permitted from the domain to each type.

Conceptually the aggregation of these domain definition lists constitutes a table, which we call the Domain Definition Table (DDT). To make the domain-type check, the TOP consults the DDT row for the executing domain, finds the column for the object's type, and compares the resultant entry against the intermediate set of access rights. Any right in the intermediate set which does not appear in the DDT entry is dropped, and the result is the final set of access rights which is transmitted to the MMU.

(Certain domains have additional, privileged, roles and may therefore obtain access rights in excess of those determined from the mandatory and discretionary checks. A discussion of this mechanism is beyond the scope of this paper.)

The above complex process cannot be performed on every access attempt. On the other hand, the checks cannot be made far in advance and saved (in a "capability," for instance), as such early binding cannot provide the access right revocation implicit in certain acl changes.

In SAT, the TOP operation load name space table (LNST) evokes the access rights check; it inserts access to a designated object at a designated segment number in a subjects's address space, and establishes the correct maximum access rights for that subject to that object. The mandatory, discretionary, and domain rights checks are performed during the execution of LNST, and then the subjects's MMU table is modified to reflect the new entry. If the LNST operation is proved to conform to the security policy and

if the MMU is proved to enforce the access rights set in the NST, the system is thereby proved to conform to the security policy for each and every instruction execution.

Domain changing may occur as a side effect of procedure call. If the called procedure is not executable within the caller's domain, either the call is illegal or a domain change is necessary to complete the call. Information concerning domain changes is stored in a Domain Transition Table (DTT), which is stored as a set of lists associated with the calling domain. The SAT system creates new subjects to handle domain changes, as required. When a call requires a domain change, SAT suspends the calling subject and activates the called subject. The called subject has a different execution context, name space, and access rights, which will prevail for the duration of the procedure's execution.

In the SAT prototype, the DDT and DTT are set at the time that a particular version of the reference monitor is installed. The number of types and domains, and the relationship between them, accordingly remains static until a newer version of the reference monitor is installed. Later versions of SAT will include facilities for the dynamic creation of types and domains.

Note that the access right computation involves the successive denial, or "crossing off" of those access rights initially allowed by the mandatory policy. This approach guarantees that omission of an access right in a DDT entry for a type, domain pair will effectively block access to that type by any program encapsulated in that domain. This guarantee is verifiable by inspection of the DDT, and provides assurance that certain types remain "private" to certain domains. Note also that it is possible to assign types to procedure objects, and place restrictions on "execute" access in the DDT. This last feature permits assurance that critical code is indeed encapsulated in protected domains. In effect, the DDT reflects, and gives assurance in, the structure of the reference monitor. This in turn permits a strong correspondence to exist between the organization of the design and the organization of the proof.

USES OF TYPE ENFORCEMENT

Implementing Integrity Policies

We would like to begin by observing that our type enforcement policy subsumes the second class of hierarchical integrity policies, that is, those in which an unchanged integrity level is bound to subjects and objects.

In order to implement a hierarchical integrity policy in SAT, it is necessary to first assign types to procedures based on their integrity level. The set of procedures possessing a given type is isolated into a distinct domain, which is the only domain from which these procedures may be executed.

Data objects are then assigned a distinct set of types, also based on integrity level. It is then trivial to devise a DDT configuration which implements the restictions of the Ring Policy or the Strict Integrity Policy.

For example, let us assume that we have three integrity levels 1,2 and 3. We would then have three types of procedures, P1, P2, and P3, (with the corresponding domains) and three types of objects O1, O2, O3. It is also necessary to have a "gatekeeeper" domain P4 for use when changes in integrity level are required.

In order to implement the Strict Integrity Policy, we need only construct a DDT configuration as follows:

Object Type:	O1	O2	O3
Domain P1:	o/m	o	o
Domain P2:	m	o/m	o
Domain P3:	m	m	o/m
Domain P4:	null	null	null

(o = observe; m = modify)

And the following DDT configuration:

Called Domain:	P1	P2	P3	P4
Domain P1:	e	e	e	cP4
Domain P2:	null	e	e	cP4
Domain P3:	null	null	e	cP4
Domain P4:	cP1	cP2	cP2	e

(e = execute and stay in current domain; cDestination = change to domain Destination.)

Tables for the Ring Policy may be similarly constructed. Note that a binding which is stated in the policy as existing between integrity levels and subjects is here mapped onto a binding between, in effect, integrity levels and procedures. This mapping is possible because the policy treats execute and observe access the same, thereby establishing a relationship between the integrity level of the subject and the integrity level of the procedure executing in the context of that subject.

The above argument shows that any set of restrictions enforceable by the second class of integrity policies is enforceable by the type enforcement policy. The first class of integrity policies, in which integrity levels of subjects or objects change, may be dismissed as impractical from the point of view of performance and proof.

Having argued that type enforcement can deal with any case that a hierarchical integrity policy can deal with, we proceed to the more interesting cases in which hierarchical integrity polices must appeal to "trust" in order to accomodate practical processing requirements.

Assured Pipelines

We will now show that the assured pipleline structure can be readily accomodated by the type enforcement policy. We will show DDT and DTT configurations based on the following types and domains:

Types: Unlabelled and Labelled data.

Domains: User, Labeller, and Output.

Unlabelled data is data which has only internal labels associated with it. Labelled data is data which is properly marked on the top and bottom of each page for output.

Unverified and possibly hostile programs are encapsulated in the User domain. The labeller module described in the previous section on assured pipelines is encapsulated in the Labeller domain and is verified to properly translate internal labels to readable form and place them in the correct positions in the data. The output module of the previous assured pipeline description is encapsulated in the Output domain and is verified to not tamper with labels. None of the domains in the example invoke any form of privilege.

The DDT which enforces the pipeline is as follows:

Object Type:	Unlabelled	Labelled
User Domain:	o/m	null
Labeller Domain:	o	o/m
Output Domain:	null	o

(o = observe; m = modify.)

And the corresponding DTT is:

Called Domain:	User	Labeller	Output
User Domain:	e	cLabeller	null
Labeller Domain:	null	e	cOutput
Output Domain	null	null	e

(e = execute and stay in same domain; cDestination = change to domain Destination.)

Note that not only does the DDT restrict the data flow, but the DTT restricts the control flow in such a manner that the pipeline must be initiated by (possibly hostile) user code in a proper manner; the Output domain is not callable from the User domain.

TYPE ENFORCEMENT AND PROOF

Factored Proofs

Assurance, in the final analysis, is based on human confidence; and confidence comes from insight and understanding. It has accordingly been a goal of the SAT project that its proofs of security be accesible to human analysis, understanding, and criticism.

This goal has led us to avoid the machine-generated proofs of previous efforts in favor of proofs which have an informally understandable underlying structure; formalism is used to permit machine-checking of our results and not as an end in itself.

We use the traditional structure of a "factored" proof, that is, an argument based on an orderly presentation of lemmas. The proof has two purposes. The secondary purpose is to convince a skeptical observer that our system is secure; the primary purpose is to give that observer insight into the precise meaning we give to the word "secure."

In order to achieve this goal we must present a proof whose organization corresponds in a fairly obvious way with the organization of the system, so that for every conclusion we draw along the way there is a clearly identified system feature which supports that conclusion. In the next section we shall outline such a proof of our example labeller pipeline.

A Factored Proof of a Labeller

The fact that a labeller is "secure" can be captured in three theorems:

Theorem 1: Only labelled information is output to hard copy.

Theorem 2: Labels are properly inserted prior to output of labelled information.

Theorem 3: Labels are not modified prior to output of labelled information.

We now present the lemmas used in our proof, and the manner in which each lemma would itself be proven.

Lemma 1: The SAT hardware properly enforces a given DDT and DTT configuration. This lemma is proven as part of the overall proof of the security of the SAT reference monitor, and is accordingly "built in" to the SAT hardware.

Lemma 2: Only the Labeller module can write to Labelled data. This lemma is proven by inspection of the DDT configuration given in the example in the previous section.

Lemma 3: The Output module will read nothing but Labelled data. Again, this is proven by inspection of the same DDT configuration.

Lemma 4: The Labeller module properly translates internal labels to external form, and inserts them at the top and bottom of each page. This lemma is proven by applying standard program proof techniques to the labeller program. The proof involves demonstrating the truth of two relatively weak assertions: that the Labeller performs a table look-up properly and that it can find the top and bottom of a page of hardcopy.

Lemma 5: The Output module does not tamper with labels. As a practical matter, this lemma will be proven using informal methods. This is because Output modules are typically complex and machine-dependent. It is accordingly difficult to capture their operation in the semantics of formal program-proof systems. Modules of this type are amenable to inspection and comprehensive testing, especially when it is known (as in this case) that their inputs come only from

formally verified code and therefore form a tractable set of test cases.

We now note the correspondence between this set of lemmas and the organization of the SAT reference monitor. Lemma 1 is a "hardware level" lemma, a global property which applies to all programs which execute on the SAT hardware, irrespective of their context or construction. Lemmas 2 and 3 are "structural" or "programming in the large" lemmas, properties which reflect the modular decomposition of the SAT reference monitor but which are not concerned with the internals of the modules themselves. Lemmas 4 and 5 are "programming in the small" lemmas, conclusions drawn about the operation of the modules which are independent of their context in the system. Thus we argue that there is a clear intuitive correspondence between elements of the system and elements of the proof.

Previous efforts to prove the security of labellers have generally been restricted to Lemma 4 and occasionally Lemma 5; that is, the proof has demonstrated that if the Labeller is invoked, then it properly labels; the proof does not demonstrate that the Labeller must always be invoked. In logical terms, the proof fails because a necessary but not a sufficient condition has been demonstrated; in design terms, the proof fails because the correctness of a module's internals has been shown but the correctness of the structure of the system has not. This situation is analogous to proclaiming a system correct when its modules have all passed unit test but integration testing has not yet been performed.

Given the above lemmas, the proof of each theorem is as follows:

Theorem 1 (Only labelled data goes out): Lemma 1 (DDT enforced) and Lemma 2 (Only Labeller writes Labelled) and Lemma 3 (Output only outputs Labelled).

Theorem 2 (Labelled data is correct): Lemma 1 (DDT enforced) and Lemma 2 (Only Labeller writes Labelled) and Lemma 4 (Labeller labels properly).

Theorem 3 (Labelled data is tamperproof): Lemma 1 (DDT enforced) and Lemma 2 (Only Labeller writes Labelled) and Lemma 5 (Output module is benign.)

SUMMARY

Hierarchical integrity policies have been shown to be inadequate to enforce the restrictions on information flow required by practical systems. An alternative policy based on types and domains has been presented which has been shown to subsume both the practical variations of hierarchical integrity polices and cases which such polices cannot handle without recourse to exceptions. The alternative is also shown to support proofs whose structure corresponds in obvious ways to the structure of the system being reasoned about.

REFERENCES

1. W.E. Boebert, R.Y. Kain, W.D. Young, and S.A. Hansohn, "Secure Ada Target: Issues, System Design, and Verification," Symposium on Security and Privacy, IEEE, 1985, 176-183.

2. K.J. Biba, "Integrity Considerations for Secure Computer Systems," The MITRE Corporation, Bedford MA, MTR-3153, 30 June 1975.

3. W.E. Boebert and C.T. Ferguson, "A Partial Solution to the Discretionary Trojan Horse Problem," these proceedings.

ACKNOWLEDGEMENTS

This effort has been supported by US Government Contracts MDA904-82-C-0444 and MDA904-84-C-6011.

ON THE LOGICAL EXTENSION OF THE CRITERIA PRINCIPLES TO THE DESIGN OF MULTILEVEL DATABASE MANAGEMENT SYSTEMS

Marvin Schaefer

DoD Computer Security Evaluation Center
Fort Meade, Maryland

Several researchers have opined that the *Trusted Computer Systems Evaluation Criteria* (TCSEC) cannot be applied to the multilevel database management problem. We do not subscribe to this view, and observe that many special-purpose but multilevel database management systems have been designed, implemented and evaluated in concert with the TCSEC. In this paper we intend to examine the nature of what has been done and, through *gedankenexperiment*, suggest the possibility of generalising on the trusted operating systems work that has been done to date.

A DATABASE INTERPRETATION

We begin by noting that every trusted operating system must necessarily implement and maintain a number of internal system tables that consistently describe the real state of the machine, its processes and its physical resources. These system tables describe and are used to control information at or about all supported sensitivity levels with respect to all of the system's subjects and objects. Those system tables that are within the TCB must be supported by functions that act consistent with the system security policy. While the TCSEC requires it be shown that the system tables are correctly interpreted by the TCB, it is also required that information flow analysis be performed on the functions that maintain or depend on the contents of the tables in order to demonstrate that the TCB does not leak classified information derived from the system tables.

In large systems, the system tables may contain several *millions* of entries. Many of the tables are updated frequently, dynamically, and asynchronously on behalf of processes and requests from all supported security levels. Such tables were characterised as a multilevel relational database in 1976 during DARPA's KVM/370 effort, and similar observations were later made about the KSOS system tables.

Although it may at first appear unreasonable to consider these tables as anything other than a highly-contrived example of a database, examination shows that they have all the characteristics of a multilevel relational database. Even if one were to argue that since the TCB implements the security level abstraction, and hence does not deal directly with classified entities, but rather with the *descriptors* of classified entities, the fact remains that the system tables constitute a database that describes multilevel data and that is used to place classified objects into the domains of untrusted subjects.

RELATIONAL DBMS ANALOGUE

A table can be likened to a relation whose *tuples* correspond to the table's rows, and whose *attributes* or *fields* correspond to the columns. The classification granularity of data in system tables may apply to fields as small as a byte or a bit, or it may be distinctly applied to larger structures, e.g., each row of a table, or in some cases to an entire table. Locating specified data in the rows of a table corresponds to *selection*, and extracting data from specific fields of these rows corresponds to *projection*. In many cases, (e.g., demand page management, I/O scheduling, etc.,) data in one system table must be correlated with data in another system table, where the data associations are given by pointer chains. This latter operation is the equivalent of the *join* operation.

The system tables can also be considered as a subschema from which "secure views" of the physical machine and its resources are derived for the user processes (untrusted subjects) consistent with the operating system's formal security policy model. Under interpretations of [B&L73], e.g., each subject's domain may be built from the space of real pages such that the read-bit is set for a domain page only if its security level is dominated by the subject's security level; and the write-bit is set only if the two security levels are equal. Under this interpretation, the TCB can be viewed as a trusted database management system: a multilevel DBMS (MDBMS) no less general than a multilevel document retrieval system or any other general purpose database management system.

SECURITY REQUIREMENTS

The TCSEC requires that each secure view of real- and virtual machine resources be implemented such that each subject be granted a set of permissible operations on its derived data view, that this data view be consistently maintained, and that the exploitation of certain information flow channels be precluded. At and above the TCSEC's B2 level, forbidden information flow channels include covert storage and, ultimately, covert channels.

It is clear that this MDBMS largely consists of trusted code, much of which should be implemented with least-privilege (B3) as trusted subjects in restricted small domains of the TCB. It should be possible to derive a constructive transformational approach to establishing the formal security requirements for each such trusted subject.

A *GEDANKENEXPERIMENT*

We propose to follow a process of "folding": that is, we begin by assuming that there exists an uninterpreted but "sufficiently secure" TCB to satisfy the B2 requirements (i.e., we hypothesize that untrusted subjects can be supported and be granted access to relatively "large" single level objects called *segments*). We recall that a Secure Relational DBMS (SRDBMS) architecture was derived by Hinke and Schaefer [RADC75] in an investigation of the possibility of designing a DBMS application system that could be used to create and maintain authorised views of multilevel databases. The project was constrained by a requirement that no modifications were to be made to any the Multics kernel code or its trusted code, and no new code was to be introduced into the privileged domains of execution (rings 0 or 1).

This leads to the idea that by using the [RADC75] architecture it would be possible to construct a completely untrusted DBMS that implements multilevel secure views of a database partitioned into single level segments and over which all accesses are controlled by the TCB while the SRDBMS implements the semantics of the database. It would then be possible to implement fully untrusted single level [i.e., least privilege] views of a multilevel internal system database of the type described above, each of whose relations is implemented in single level segments. This would of course be illegitimate in both theory and practise, since this database would have to have already been implemented within the TCB in order to support our hypothetical untrusted database implementation.

However, suppose for the moment that this implementation were legitimate. Then once having done it, we could proceed to define the precise semantics and the permissible operations that could be performed on such a database on behalf of an untrusted subject operating at an arbitrary formal security level. Among the defined operations, we most have fully detailed the required procedures and the conditions that must hold in order to update the individual components of a multilevel view, to add or delete a tuple, and to view a tuple.

Operations that Must be Trusted

In some cases the [RADC75] architecture requires that an operation be performed in quanta that are initiated at several distinct security levels. (For example, in order to update (or delete) a multilevel tuple with purely untrusted code, direct application of the Simple Security Condition and the *-Property require that the fields in a tuple be sequentially updated (deleted), each field in turn by a subject that acts at precisely the security level of the field undergoing modification. Such a regimen is not only awkward, but it introduces a number of potential semantic integrity problems in the face of concurrent readers and updaters on the system who could be operating at different security levels.) These cases arise precisely because of the lack of a trusted process and/or a trusted path in the [RADC75] paradigm's use of an underlying security kernel. We conjecture that such analysis leads directly to the identification of the set of database-specific operations that need to be supported by a trusted implementation: it is the set of operations that cannot be completed under the [RADC75] syndrome at a unique security level.

This makes it possible to produce a precise definition of the data-specific input and output assertions and constraints that would be required to define a trusted process that would perform the same operation as an atomic state transition. Given such a precise characterisation, we would suggest that the untrusted sequential operation could be replaced by a trusted, but suspicious, subject that would be invoked by a kernel call. Assume then, that the kernel call and the trusted subject exist, replacing the untrusted code sequence by the kernel call and its trusted implementation. Continue to proceed along these analytical lines until such time as no remaining non-atomic database operations need be performed as atomic operations.

At this point, we claim, the MDBMS has been partitioned into its trusted and untrusted components, completely along the lines of the TCSEC's architectural and assurance requirements. However, since the databases that were first modeled as partitioned relations were integrated into a heterogeneous classified single database, and since the databases were derived from the hypothesised underlying components of the TCB, we have managed to completely define the internal structure and semantics of the TCB's hypothesised underlying MDBMS schema. We would further observe that by following the method of our *gedankenexperiment* to derive the trusted MDBMS primitives, we obviated the need to store the multilevel database in single level segments. It appears that in this way each of the TCB's required system table implementations can be constructed with the required assurances.

REFLECTIONS

We are sensitive to the fact that the thought process we followed led us to construct a one-of-a-kind trusted database management system. Although we were capable of identifying the primitive trusted DBMS functions needed to support this application, we were not able to come up with a closed-form solution to the general problem. We worked with a database wherein it was possible to define the precise classification of every data *relationship* at the time the database was conceived. While the approach explicitly allows for the dynamic classification of new entries in predefined relations, we further constrained the generality of our result by limiting the set of permissible domains for join operations, thereby not having to address the problem of deriving classifications for dynamically-created relations. However, we harbour no doubts that archetypes of the trusted primitive functions we derived under this process would be present in a wide variety of trusted database management applications.

In reflecting on the simplification we used of the general trusted database management problem we employed in this *gedankenexperiment*, we recall several observations on security policy models that came from the Air Force Summer Study on Multilevel Data Manage-

ment Security [NAS83]. Some observers have said that the Bell and La Padula Model does not apply to the MDBMS problem. The reasons given have ranged from the misconception that the [B&L73] does not apply to "small" objects where the granularity of classification is "fine", to the identification of a lack of detail on the treatment of classified entities that contain subentities that are distinctly and individually classified.

It is to be observed that classification in a database is rarely *determined* purely on a *syntactic* basis, but rather on the basis of an *association* between specific data entities. Hence, there are contexts in which a specific data value, e.g. 17.3, may be viewed as a TOP SECRET value while 42 would be CONFIDENTIAL, and there are contexts like this paper where both numbers are unclassified. Such *semantic* bases for classification must either be based on a trusted implementation of an approved dynamic algorithm or they must be determined by a trusted path communication with an individual possessing original classification authority.

We are aware of some classification decisions that are so complex that no such algorithm has been produced, while we are aware of simpler cases in which algorithms can be formulated. In either case, it appears evident that a classified record cannot be entered into a database until it and its constituent classified entity classification relationships have been identified and assigned. The wording of the preceding sentence hints at the potential complexity of a complete classification determination and assignment. It is not only the case that the components of a new tuple need to be classified, but it is also necessary to consider the classification of all possible join operations in which the tuple could be involved, including those potential joins between the new tuple and tuples in *different* relations.

If such algorithms can be expressed, we see no reason to prevent their implementation as trusted subjects under the constraints of [B&L73]. If the algorithms cannot be expressed (e.g., in cases where classification is determined intuitively), then a trusted subject would necessarily need to be invoked through a trusted path in order to allow an authorised individual to communicate all of the classification requirements to the MDBMS.

In both cases it would be required that assertions be formulated and proven to demonstrate that the MDBMS preserves the invariance of secure state for the system. This clearly calls for a precise definition of the semantics of each of the MDBMS operations with respect to the domain of each security-relevant operation. Potentially, every modification to the value of any tuple would require a determination of the classification of the tuple and all of the identified classified semantic data interrelationships mentioned above.

It may serve as a useful example to observe that volume III of [B&L73] treats the classification of the relationship between two classified objects. The compatibility requirement for a directory and segment hierarchy requires that all paths from the root node to a directory or segment be monotonically non-decreasing in classification level. Since the hierarchy is represented as a directed graph whose nodes are objects (single level directories or segments), it can be observed that the individual arcs between compatible hierarchy elements are classified. This is the equivalent of assigning a classification to a join operation between pairs of elements in the relation that represents the hierarchy. (A directed graph can be represented by a relation in which a predecessor and a successor field is defined for each element.) Node x is the immediate predecessor of node y just in case there exist domains *pred, succ* such that

$$pred(y) = x \text{ and } y \neq succ(x).$$

This is precisely an example of a classified join operation. Every modification to the hierarchy must be shown to preserve compatibility. In the Multics implementation of [B&L73], the operations on upgraded directories and the establishment of links are equivalent to modifying the arcs in the graph of the hierarchy and are necessarily implemented with trusted code.

We believe that the methods used in previous trusted operating system development provide useful insight into the MDBMS problem and to its partial solution. Our investigation suggests that it is possible to provide adequate definitions and constraints for the trusted subjects needed to implement an MDBMS only if the complete semantic classification requirements are specified for each database under consideration.

We recognise that much of the code needed to support a multilevel database management application may need to be trusted, and would observe from the experience of [RADC75] that the preponderance of trusted code would be required for dynamic databases in which classification is semantically derived than in those that are only referenced and wherein the data classification relationships can be derived syntactically.

ACKNOWLEDGMENTS

We would like to thank our colleagues D. Elliott Bell, Earl Boebert, Swen Walker, Brian Hubbard and other researchers in the Center for their helpful and critical comments on our *gedankenexperiment*.

REFERENCES

[B&L73] D.E. Bell and L.J. La Padula, "Secure Computer Systems: Mathematical Foundations;" "A Mathematical Model; and "A Refinement of the Mathematical Model," MTR-2547, vol. I, II, and III, The MITRE Corporation, Bedford, MA, March, November and December 1973 (also ESD-TR-73-278, vol. 1-3.)

[NAS83] Marvin Schaefer, chairman, "Multilevel Data Management Security," Committee on Multilevel Data Management Security, Air Force Studies Board, National Academy of Sciences, Washington, DC, 1983.

[RADC75] T.H. Hinke and Marvin Schaefer "Secure Data Management Syste[illegible] RADC-TR-75-266, Rome Air Develo[illegible] Center, Air Force Systems [illegible] Griffiss Air Force Base, November 1975.

PANEL

ON

THE NATIONAL TELECOMMUNICATIONS

AND INFORMATION SYSTEMS SECURITY COMMITTEE

The panel members represent the NTISSC's Subcommittee on Automated Information Systems Security (SAISS).

Panel Chair:

Catharine M. Martinez
SAISS Executive Secretary

Panel Members:

Dr. Robert L. Brotzman
SAISS Chairman

Mr. James Burrows
SAISS Department of Commerce
Representative
Chairman, SAISS Working Group for
Information Systems Security
Criteria, Standards and Definitions

Mr. Lynn McNulty
SAISS Department of State
Representative

On September 17, 1984, the President signed National Security Decision Directive 145, "National Policy on Telecommunications and Automated Information Systems Security." This directive broadens the Center's responsibilities from the DoD level to the National level, and it also establishes a government-wide committee to guide the conduct of national activities directed toward safeguarding telecommunications and computer systems.

This committee, the National Telecommunications and Information Systems Security Committee (NTISSC), is composed of representatives from over 20 government departments and agencies. Its primary responsibilities are to develop specific operating policies and provide guidance on telecommunications and automated information systems security to the departments and agencies of the federal government. Two subcommittees -- one for telecommunications security and one for automated information systems security --report to the NTISSC. The Subcommittee on Automated Information Systems Security has already, within its first ten months, submitted to the NTISSC its first annual evaluation report on the status of automated information systems security in the federal government. This subcommittee is currently developing standards on password usage and environment guidelines; developing security criteria for personal and shared computers; and drafting a program to encourage, advise, and assist the private sector in developing computer security products. The subcommittee is also drafting National Telecommunications and Information Systems Security policies and directives on: definition for 'sensitive, but unclassified, national security related' information and for 'critical' systems; Automated Information Systems Security Awareness Program; and, Automated Information Systems Security Education and Training Programs.

EDUCATION AND AWARENESS PROGRAMS IN THE NATIONAL ARENA

Cathy Hanks Thomas
DoD Computer Security Center
9800 Savage Road
Fort George G. Meade, MD 20755-6000

As an understanding of the computer security threat becomes more widespread in Government circles, and as the technologies to counter that threat become more available, the need for comprehensive national-level education and awareness programs becomes more urgent. This paper provides a general description of what should be done to respond to this need by answering three basic questions: Who is the target population of these programs? What kinds of activities are needed in these programs? And, who is going to conduct these activities? I will also describe some activities being conducted by the DoD Computer Security Center in the education and awareness areas and will then conclude by suggesting what is not being done, but should be.

Before I begin, however, a word about definitions. Different organizations use the terms "education," "training," and "awareness" in different ways. In our program at the Center, and in this paper, the term "education" refers to courses, briefings, workshops, and other scheduled activities in which individuals learn about computer security. These activities traditionally are conducted by training organizations. The term "awareness" refers to an initial orientation for new employees to good security practices and periodic reminders. This type of program is traditionally conducted by security organizations.

Now to the first question: What is the target population of Federal Government-wide education and awareness programs? The target population can be categorized by the roles or functions of individuals. We have identified four types of workers: the general user, the manager, the security or education specialist, and the technical computer security professional. Each of these groups has different needs for education and awareness. The general user category includes anyone operating, accessing, or otherwise coming in contact with a computer. This may be a clerical employee using a word processor, an operator on a large mainframe, or a supervisor accessing a network through a personal computer. This very large population is the primary focus of an awareness program. The second group, managers, needs to be addressed not only by an awareness program, but also may need courses, seminars, or workshops to get a more detailed understanding of the principles of computer security. This more detailed knowledge is needed to support decisions that impact on the implementation of computer security principles. In the third group are the security or educational specialists involved with computer security programs. This group needs in-depth training on the tenets of computer security, with some technical detail, and can also benefit from activities, such as workshops with other educators, on how to establish and conduct a good computer security program. At these workshops, for example, there can be an exchange of those all-important "lessons learned" and the initiation of procedures and contacts to share resources. The fourth major group is composed of the computer security or computer science technical specialists. These individuals need very specialized courses on the techniques of improving computer security by both hardware and software.

A very obvious characteristic of the target population is its size. Within the Executive Branch alone there are some 85 unsubordinated organizations. This number counts each Cabinet-level department as one organization. These Cabinet-level departments are made up of many subordinated organizations. Within the Department of Defense, for example, not only are there the four military services but also 12 defense agencies and the world-wide field activities of the Office of the Secretary of Defense. Some of these organizations within the Executive Branch are large - like the Department of Defense - and some much smaller - like the Nuclear Regulatory Commission. But most organizations within the Executive Branch, even if they are small, have a real need for computer security education and awareness programs. Whether their information is covered in NSDD-145 (and hence are of concern to the Center) or not, organizations such as the Tennessee Valley Authority, the Agriculture Department, OSHA, and the Panama Canal Commission have a requirement for computer security and for activities to educate their employees.

How many people are we talking about? The total number of people in the Federal Government that need some kind of exposure to computer security education and awareness is staggering. The accompanying chart (Figure 1) reflects approximate figures for the number of employees in various categories and the number of these that we estimate need either a formal course in computer security (or at least computer security covered in some formal training), or a training session tailored to their individual work environment, or should be the target of an awareness campaign.

Let us move on to the second question: What kind of activities are

needed in education and awareness programs for these individuals? In the area of awareness, two types of activities are needed: initial indoctrination and then reinforcement. The initial indoctrination should include an explanation of what the threat is and the security practices the employee should follow. The responsibility of every Federal employee to protect - and respect - information must be emphasized. In many agencies, especially those that deal with classified or sensitive information on a regular basis, this indoctrination to computer security can be a part of the indoctrination into the broader subject of information security. The related issues of, for example, computer security, physical security, communications security, and procedural controls can be discussed in what can be termed a "holistic" approach to security. At my agency, for example, this can be done when the employee receives the badge that allows entrance into the building compound.

This initial indoctrination must be followed by periodic reinforcement activites. These activities can take many forms; posters, videos, participation in "security weeks," and articles in newsletters are some suggestions. The materials or events should repeat the message heard in the initial indoctrination and serve as daily reminders of the employee's responsibility.

The components of an education program are also varied. Formal courses are used most frequently. To satisfy the needs of the target population a number of different kinds of courses are required. Each organization should design a curriculum appropriate to the needs of their employees, the particular environment in which they work, and the degree of security needed. These are some types of courses that should be considered:

- an introductory course for those who only need a general overview of the tenets of COMPUSEC;
- a course geared to security professionals who need to integrate COMPUSEC information into programs on physical, communications, and personnel security;
- a course for managers to address acquisition requirements, establishment of procedural controls, etc.; and
- technical courses for computer specialists who are implementing safeguards on the systems.

If sufficient resources are available and the need exists, an agency may want to consider a course in the legal and ethical aspects of computer security.

A comprehensive education program should also include training sessions tailored to a specific office or function. At the Center, we call ours the "Roadshow." It is a multi-media approach to providing information on computer security to an organization at their site. Using instructional modules, the roadshow can range from a 1-hour to a several-day presentation, covering topics such as the threat to information in computers, risk assessment, standards for evaluating security, and software and hardware available to improve security.

FIGURE 1

APPROXIMATE SIZE OF TARGET POPULATION

	TOTAL NUMBER OF EMPLOYEES	INCLUSION IN FORMAL COURSE	TAILORED SESSIONS	AWARENESS ACTIVITIES
EXECUTIVE BRANCH (civilian)	2,830,000	142,000	283,000	1,415,000
MILITARY SERVICES				
OFFICERS	265,000	265,000	26,000	132,000
ENLISTED	1,651,000	825,000	164,000	330,000
LEGISLATIVE, JUDICIAL BRANCHES	55,000	3,000	6,000	28,000
TOTAL:	**4,801,000**	**1,235,000**	**479,000**	**1,905,000**

This brings us to the third question, one that generates some controversy: Who should conduct these activities? The immense size and diversity in functions of organizations in the Government suggest that it is not very useful to talk about national programs for education and awareness. Rather, we must think in terms of a national-level effort to insure that each organization within the Federal Government has adequate computer security education and awareness programs. For large agencies this may mean a full-fledged program with courses developed and conducted in-house. Smaller agencies may have only an awareness program and rely on other sources for formal courses.

For example, some organizations already have active, successful in-house education and awareness programs established, such as the Department of Energy. They have courses, a newsletter, and many other activities that are the hallmark of a good program. However, for formal courses, some organizations, especially smaller ones, are looking to Government training organizations to provide the formal courses and more tailored training sessions. The DoD Computer Institute and the Office of Personal Management both include computer security courses in their curriculum, as does the Center. Computer security courses are also available at some academic institutions. While these are often geared more to the theoretical approach than specifics on how to do COMPUSEC in an operational environment, they should not be overlooked as a valuable source of education. And, of course, there are the courses available from the private sector, both the for-profit firms and the non-profit organizations, such as professional societies. These are offered both on a regular basis as scheduled classes and as a part of conventions.

In support of the need for education and awareness programs in the Federal Government, the DoD Computer Security Center sponsors a number of activities. As previously mentioned, we conduct a number of formal courses. These range from a national-level course for managers to very intensive technically-oriented training for evaluators of products and systems.

Also mentioned earlier was our "roadshow." We are prepared to travel to a requesting agency's location and present an orientation to computer security tailored to meet that agency's particular needs.

We are assisting some agencies in establishing new education or awareness programs. This assistance includes the sharing of our own resources, such as course outlines, briefings, graphics, and videos, and assisting the organization in identifying resources that may be available from other organizations, such as OPM.

We are currently working with the Treasury Department in setting up what we hope will be a model awareness program. From our experience at Treasury we hope to develop a "handbook" on how to establish such a program.

Another major effort is to make a wide selection of products available, at little or no cost. We are producing, for example, a quarterly security awareness poster. Two video tapes on access control have been among our most successful products to date. They were produced to explain to a non-technical audience why access control is important. We have distributed, in just 3 months, over 500 copies of these tapes. The next video that we will produce is on the Center's Criteria for trusted systems, the Orange Book. This tape will be available in late spring. If funds permit, we also plan to produce, during this fiscal year, a video on password management and one which will serve as an orientation to computer security for the executive.

In support of what we see as our role as a clearing house on information, we are in the process of setting up several data bases on education-related information. Listings from the data bases will be widely distributed, and eventually we hope the data bases will be part of a Center electronic bulletin board available to the public. The first data base is on educational activities. We are including courses, workshops, and seminars offered by Government agencies, by academic institutions, and by the private sector. The data base is nation-wide in scope. If you have activities that you would like listed in the data base, please contact us. Inclusion in the data base does not imply endorsement or recommendation of the course by the Center. We have wrestled with this issue; it would be nice to be able to evaluate each course. However, questions about what a course should include and how to evaluate training seem very complex. We hope to address this issue one day, but do not want to delay establishment of the data base while we develop an evaluation procedure.

By next spring, we hope to have a data base on upcoming conventions and conferences and one on computer security education and awareness products. Again, we will make every effort to see that any organization or company that wants to have data input into the data bases has that opportunity.

One activity that we hope will have strong impact on education and awareness in the Federal Government is a symposium that we are sponsoring on October 30th for computer security educators in the Federal Government. We are meeting to learn about each others' programs and to discuss how we can help each other, especially in sharing resources such as course materials. We hope that this becomes an annual event, increasing both in size and

length from this year's modest, first effort.

One very basic difficulty that seems widespread in the Government is that employees are entering the work force having had no previous exposure to computer security. This is especially true of the more technical aspects of implementing security. It is great when a manager finally understands the threat and wants to do something about it, but we find that when most managers go to their ADP programmers or system designers and say "give me some security," the ADP specialists don't know how to do that.

One way we are addressing this problem is by doing what we can to encourage academic institutions to address computer security. For example, discussions have been held with the University of Maryland on the possibility of offering a Master's degree in computer security. Some individuals in academia share our concerns. For example, Professor Janet Cook at Illinois State University is working on exercises that can be incorporated into existing courses. A course on designing operating systems, for example, would include incorporating security into the design. Professor Cook believes that this must be done at the undergraduate level. She suggests, for example, that programmers must be taught to include security from the very beginning. This goes along with the DoD Computer Security Center's tenet that security must be designed into a system, not added on later.

We also will be encouraging the inclusion of computer security awareness in management courses. Educational institutions, both government and private, are recognizing that managers need to be "computer literate." We believe that being literate should include an appreciation of security requirements.

In all our education and awareness activities at the Center, we follow a philosophy of openness and sharing. All the education and training material that we develop is available for distribution to any organization. We are trying to "package" our resources to facilitate this sharing. Often this sharing can be done at little or no cost to the receiving agency. For example, to receive a copy of our video tapes, we ask only that an agency forward a blank tape.

Lastly, there is the problem of what is not being done, but should be. Mentioned above is the situation of employees entering the work force with no knowledge of computer security. Actually, the problem is worse than that: many new employees have learned the wrong practices and attitudes. The average American today does not understand that computer resources and the information stored or manipulted in them is property. Thus the hackers are unconcerned about gaining unauthorized access to data. But the problem also emcompasses sharing passwords or posting them on the side of a terminal, using a computer at work for personal business, copying licensed software, and other unauthorized actions. We need to start an education process long before the employee joins the workforce. We need to start this process before the employee reaches college. Grade school is probably a good starting point - although it now seems de rigueur for pre-schools to have a PC. Teachers and, just as importantly, parents should be emphasizing the responsiblity that every computer user has to protect his or her own data and respect the data of others. Teenagers who would not dream of entering an unlocked house and taking valuable items think that gaining unauthorized access to a data base and copying data is a game. Responsiblity and respect should be taught right along with instruction on how to use computer resources.

So who should do something to correct these attitudes? The answer to that is almost all of us. Some suggestions: Do you belong to a professional society that should be addressing the issue of security at a meeting or conference? If you know something about computer security, why not offer your expertise to your local school system as a guest speaker? Why not recommend to your local school board that material on responsbility and respect of computer resources be included in the schools' cirricula? Are you a parent? Remember that education begins at home..

A national-level effort in the area of computer security education and awareness is a commitment of the DoD Computer Security Center. The programs outlined in this paper are a beginning. But we must all work together or we all deserve the consequences of a workforce that remains unaware.

NO HARM INTENDED:

A BEHAVIORAL ANALYSIS OF YOUNG HACKERS

Julie A. Smith

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

NOTE: The ideas which are presented in this paper are those of the author and cited references and do not necessarily reflect the views of the DoD Computer Security Center.

ABSTRACT

The actions of young hackers are potentially dangerous even though such hackers often obtain unauthorized access to computer systems with no conscious intent to do harm. This paper provides a behavioral analysis of young hackers' actions which describes them as being the result of interaction between certain personality characteristics and strong social and environmental influences within the hacker culture. This analysis is then applied to suggest new approaches to decreasing hacking among young people.

INTRODUCTION

Society's attitude toward hackers is currently an ambiguous one. On the one hand, people admire their expertise and their sense of rebellion. They see hackers as being able to conquer a powerful force in everyone's lives by manipulating computers and the people who run them. Yet, at the same time, people are disturbed by their technical prowess. The hackers' unauthorized entries into databases challenges other people's right to privacy and may even threaten the security of the nation at times. In addition, the hackers' ability to manipulate computers can become very frightening for most non-hackers when they realize that, because they do not possess such skills or the aptitude for them, they will never have as much control as hackers do.

The press generally uses the term "hackers" to refer to people who break into computer systems. Often the hackers' world is described using a sinister tone. An illustration of this is given by the Washington Post in the following description that it gave of hackers' antics as portrayed by a writer in Newsweek:

> It was a terrific story, and Richard Sandza knew it. In four columns, which his editors at Newsweek headlined "Night of the Hackers", Sandza led the reader through the disembodied and mostly illegal world of bright young men who play a kind of cross-country electronic chicken with their home computers. Using long-distance telephones that they break into by duplicating telephone company tones, . . . the hackers log onto each other's underground "bulletin boards" to trade surreptitiously obtained corporate telephone numbers and passwords, or post valid credit card numbers, or carry on silent computer-screen conversations at hours when good high school students are supposed to be in bed.[1]

The picture that is presented by the above passage may not necessarily be a totally accurate representation of the entire hacker world: some authorities assert that the type of access which the vast majority of hackers have to computer systems has not resulted in major computer crime. Such access usually consists of browsing through files and sometimes changing or destroying information. In fact, such authorities add that insiders who are knowledgeable about their computer systems often pose a much greater threat to computer security than hackers do.[2] These assertions are difficult to prove statistically; society may not yet know the extent of the damage which hackers cause. Until reliable methods of assessing the value of information resources and reporting losses are instituted throughout business and industry, the accurate measurement of damage will remain difficult.

Thus, although it cannot be fully determined how much damage is done through hacking, one aspect of hacker behavior can be stated with reasonable certainty: it appears as if hackers can be divided into two groups, those who break into computer systems with the idea of doing damage and those who enter computer systems with no conscious intent to do harm. The motives of the former group are often easily discernible. For example, such hackers may obtain unauthorized access to systems for personal financial gain, or they may be disgruntled employees who seek to get revenge on their former or current employers by causing damage to their systems. In contrast, the latter group's motives are much more difficult to determine. Hackers who do not intend

---

[1]Cynthia Gorney, "Hack Attack: Computer Whiz Kids Harass Magazine Reporter," The Washington Post (Dec. 6, 1984), p. B1. To read the article which is described in this passage, see Richard Sandza, "The Night of the Hackers," Newsweek, 104(Nov. 12, 1984), pp. 17-18. Sandza later received threatening phone calls from a small group of irate hackers after they read his article. That group also posted his name, address, and credit card numbers on an electronic bulletin board. For that story, see Richard Sandza, "The Revenge of the Hackers," Newsweek, 104(Dec. 10, 1984), p. 81.

[2]Tekla S. Perry and Paul Wallich, "Can Computer Crime Be Stopped," IEEE Spectrum, 21(May 1984), p. 34.

to harm computer systems are often computer enthusiasts in their teens or early twenties[3], and they usually see nothing wrong with their actions. In many cases, it is not until authorities start knocking on their doors that they begin to question what they are doing, and even the presence of the law does not bother some young hackers. Yet, whether they realize it or not, they sometimes cause trouble by manipulating data and break laws by misusing the telephone. Also, because they are so willing to share information about their exploits with other hackers, the potential exists for unfriendly interests to take advantage of their talents.

The conventional approach to dealing with young hackers who enter systems with no wish to cause damage is to try to teach them about computer ethics. So far, this technique has had only mixed success.[4] Perhaps what is needed are approaches that include the moral implications of computer usage, but that also extend beyond ethics. The purpose of this paper is, first, to provide the background for such approaches through a behavioral analysis of young hackers that:

1. shows how the personality characteristics of young hackers attract them to work with computers and also play a role in their initiation into the hacker culture, and

2. demonstrates how strong social and environmental influences within the hacker culture reinforce hacker behavior.

In addition, the above knowledge will be applied to suggest effective ways to change hacker behavior among young people.

WHO ARE YOUNG HACKERS AND WHAT DO THEY DO?

Not all young hackers who enter systems with no intent to do harm are involved in exactly the same type of hacking activities. Probably the best illustration of this fact is provided by a hacker who puts such hackers into several categories according to what they do. First, there are the "Novices." These are hackers in their early teens who do not see hacking as a serious activity. They treat it as if it is a game, and, as a result, they may sometimes threaten to do harm to a computer system. However, such threats are usually idle ones. Because they are just beginning to gain knowledge about computers and hacking, they often get their information about how to break into systems from more experienced hackers.[5] The second group is the "Students." These young hackers are especially interested in the educational aspects of hacking. Once they gain access to a system, they tend to work on it as often as possible in order to learn as much as they can about that system. They are definitely the most serious-minded group. The third group, the "Tourists", are young hackers who spend most of their time finding creative ways to gain access to systems. However, once they get inside a system, they do no further exploration of it.[6]

Interestingly enough, the population of young hackers who engage in the above activities is almost one hundred percent male.[7] One possible explanation as to why there are so few female hackers is that it is difficult for young girls to become involved in the computer culture. The primary way in which most children are initiated into the computer world is through video games, many of which involve the use of highly-developed spatial abilities. This puts young females at a distinct disadvantage, since numerous studies have shown that males excel in such skills.[8] Also, the fact that young females tend to avoid demanding situations while young males learn to deal with them adds to this difference in ability between the two sexes.[9]

Other explanations for the lack of female hackers include the suggestion that the levels of aggression within the hacker culture make it unattractive to females in the first place. Also, when females are having problems with interpersonal relationships, they are less likely than males to use a relationship with a machine as a substitute for a relationship with a person. Thus, they would be far less likely to become involved in the hacker culture.[10] These last two theories may be correct, given that extensive psychological research over several decades has found that males are consistently more aggressive than females,[11] and that females, in comparison to males, tend to be much more concerned with the interpersonal aspects of situations.[12]

---

[3]Bill Landreth, Out of the Inner Circle: A Hacker's Guide to Computer Security, (Bellevue, WA, Microsoft Press, 1985), p. 59.

[4]Dennis A. Williams and Richard Sandza, "Teaching Hackers Ethics," Newsweek, 104, Jan. 14, 1985, p. 76.

[5]In an examination of several electronic bulletin boards and "underground" hacker newsletters, it was found that hackers who use other hackers' information to gain access to systems are often referred to as "cookbook" hackers since they are merely following a recipe for entry.

[6]Landreth, pp. 61-67.

[7]Sherry Turkle, The Second Self: Computers and the Human Spirit, (New York, Simon and Schuster, 1984), p. 210.

[8]Sara Kiesler, Les Sproull, and Jacquelynne Eccles, "Second-Class Citizens," Psychology Today, 17(March 1983), p. 46.

[9]Patricia F. Campbell and George P. McCabe, "Predicting the Success of Freshmen in a Computer Science Major," Communications of the ACM, 27(Nov. 1984), p. 1113.

[10]Turkle, p. 210.

[11]Kay Deaux, The Behavior of Men and Women, (Monterey, CA, Brooks/Coles Publishing Company, 1976), p. 82.

[12]Ibid., p. 105.

PERSONALITY CHARACTERISTICS

Many of the skills that are used in hacking are learned from computer programming. Thus, a look at the type of people who enjoy using computer programming skills provides a general picture of the basic hacker personality. One of the most popular vocational interest tests given, the Strong-Campbell Interest Inventory, groups occupations into six major themes based on the types of activities which people do within their jobs. Of course, many occupations touch on more than one theme. According to this test, computer programmers are described as being mainly "investigative" in nature:

> This theme centers around science and scientific activities. Extremes of this type are task oriented; they are not particularly interested in working around other people. They enjoy solving abstract problems, and they have a great need to understand the physical world. They prefer to think through challenges rather than act them out.[13]

Further analysis by the Strong-Campbell Interest Inventory reveals that computer programmers also fall into the "conventional" theme. This theme describes people who prefer highly-ordered activities, and this relates directly to the fact that the computer languages which programmers use have definite structure to them.[14] As a result of the work being so structured, it is important for computer programmers to be logical thinkers who are patient and persistent and whose work is accurate even under pressure.[15]

Thus, computer programming work is the perfect medium for a person who has both the "investigative" and "conventional" traits: although computer languages have rules, the process of computer programming is a highly creative one in that a programmer can choose to approach a problem from a variety of directions. As a result, it is difficult if not impossible to find two programmers whose solutions to the same problem are written in identical software code.

The description of hackers as computer programmers can only be taken so far, for the fact remains that not all computer programmers are hackers. The key to what differentiates the hacker from the non-hacking programmer lies in the idea that hackers are extremely bright, highly-driven, and very competitive people who become addicted to the creative element of the computer programming process. As Turkle states: "What sets them apart is that they work for the joy of the process, not for the product."[16] Even the very origins of hacking show this: the start of this phenomenon lied in the Signals and Power Subcommittee of the Tech Model Railroad Club at MIT in the 1950's. This group of male undergraduate students was intrigued with the complex relationships that were formed by the maze of wires, relays, and switches which enabled their monstrous model railroad system to run. In the end, the group applied their fascination to similar complex relationships within computer hardware and software. They were definitely in the right place at the right time: they were able to work on one of the first PDP-1's ever built by the Digital Equipment Corporation, and they also had the opportunity to learn computer programming under John McCarthy, who is considered one of the three founding fathers of artificial intelligence.[17]

At MIT, the term "hack" had long been used as a name for any of the crazy college pranks that students often pulled. However, the Signals and Power people used the word with a slightly different emphasis. To them, it described "a project undertaken or a product built not solely to fulfill some constructive goal, but with some wild pleasure taken in mere involvement."[18] They gave the word a sense of importance by emphasizing that "innovation, style, and technical virtuosity" must be characteristics of an act in order for it to qualify as a "hack" and for the person who did it to be called a "hacker."[19]

For young hackers, the creative process of working with the computer presents a mental challenge. They thrive on being able to continually learn something new. This relates directly to one of the personality traits of hackers which was described previously: they are curious about the workings of the physical world. In fact, one hacker suggests that, because most hackers are very bright, they are bored with school, and, as a result, they look to hacking for the intellectual stimulation which their everyday educational environment is not providing.[20]

In addition, hacking provides them with a way to deal with their need for power, control, and certainty through the use of structure, and, as a result, they become totally immersed in the machine. In order to explain this concept, Joseph Weizenbaum refers to hackers as "compulsive programmers" and likens them to compulsive gamblers, because he views both compulsions as being rooted in dreams of power.[21] With every successful entry that they make on the computer, hackers see themselves as asserting authority over the machine and/or whoever is connected to it. This causes them to constantly strive for interaction with the computer in order to experience the feeling of power which comes with that sense of control. Thus, they are more concerned with gaining this sensation of power than they are with the effects of their actions on

---

[13]Strong-Campbell Interest Inventory, Form T325, distributed by National Computer Systems, Inc., Minneapolis, MN, 1981, p. 2.

[14]Ibid., p. 2.

[15]U.S. Department of Labor, Bureau of Labor Statistics, Occupational Outlook Handbook (Washington, D.C., April 1984), p. 179.

[16]Turkle, p. 204.

[17]Steven Levy, Hackers: Heroes of the Computer Revolution (Garden City, New York, Anchor Press/Doubleday, 1984), pp. 7-8.

[18]Ibid., p. 9.

[19]Ibid., p. 10.

[20]Landreth, p. 59.

[21]Joseph Weizenbaum, Computer Power and Human Reason: From Judgment to Calculation (San Francisco, CA, W.H. Freeman and Company, 1976), p. 124.

others. This is quite unlike the behavior of non-hacking computer programmers: non-hackers use the computer in order to address a problem which is to be solved rather than seeing a problem mainly as an opportunity to interact with the computer.[22] As a result, non-hacking computer programmers are far more likely to design programs in their entirety before coding them, while hackers often work with no real sense of a long-term goal. They tend to dabble in one section of a program or system until they become bored with it; then they move to another section. They do not plan ahead.[23]

Evidence of a strong need for control is seen in the power struggle which young hackers see themselves as having with any type of authority who has the capability to take the source of their enjoyment, hacking, from them. Indeed, negative references to authorities are found throughout hacker-related literature. For example, in one issue of an "underground" hacker newsletter, the appearance of AT&T's logo (see Figure 1 below) which hovers above an outline of the earth on their new credit card is described as being similar to "The Death Star" which the evil Darth Vader rules in the popular movie Star Wars.[24]

Figure 1: AT&T's Logo

The fact that many parents, teachers, and law enforcement officials are sadly lacking in their basic knowledge of computers does not help this situation. Indeed, even the Federal Bureau of Investigation uses training and equipment that is at least ten years too old to fight computer crime.[25] Young hackers find it difficult to respect those who know nothing about their favorite subject.

SOCIAL AND ENVIRONMENTAL REINFORCEMENTS

It has been suggested that hacking is a stage through which young hackers pass.[26] Once they leave college, they usually settle into a mode where they work eight or more hours a day in order to support themselves just like everyone else, and they cease their hacker activities. They tend to select computer-related occupations and are often dependable and knowledgeable employees.[27] Some young hackers even use the experience that they acquired through their hacking activities to find employment upon graduation from college, as one hacker explains as he describes the type of hacker which he calls a "Student":

> A Student often roams undiscovered on your system until he walks in looking for a job. When you see his resume, you will find that he's had three years' experience on the same computer you have, doing the same type of programming you need done. Strange, but somehow Students seem to know just the person you want to hire...and when.[28]

The absence of hacker behavior beyond college which is described above is best explained by the fact that, when they leave college, young hackers abandon strong social and environmental influences which interact with their personality characteristics to reinforce hacker behavior. In fact, recent clinical research and experience demonstrates just how important these social and environmental influences really are. In particular, the research involves work with people who suffer from "technostress", which is the development of "aberrant and antisocial behavior" as a result of an "inability to cope with the new computer technologies in a healthy manner."[29] One psychologist has found that an effective way to help those who have the "technocentered" form of this illness, that is, those whose maladaptation consists of too close of an identification with computers, is to change the social and environmental factors in their lives.[30]

For young hackers, the process of socialization into the hacker culture demonstrates well what the social and environmental reinforcements of their behavior are. The first step in this process is a loss of interest in academic subjects outside the realm of computer science. Because of this, a drastic drop in grades is often seen. Secondly, developing hackers begin to adjust certain environmental factors, such as their schedules and

---

[22]Weizenbaum, p. 116.

[23]Ibid., p. 118.

[24]"Death Star Cards Spell Woe," 2600, 1(Feb. 1984), p. 3. 2600 is generally seen as an "underground" newsletter, although, in one of its issues, this is denied: "We are not an 'underground' magazine; we don't break laws or publish items that are illegal to publish. We simply discuss interesting things that can be done with today's technology." (See "2600 Writer Indicted," 2600, 1(June 1984), p. 3.) The newsletter's name is a shortened form of 2600 Hertz, which is one of the frequencies that is used in illegal devices that allow users to make free long-distance telephone calls.

[25]Tom Shea, "The FBI Goes After Hackers," Infoworld, 6(March 26, 1984), p. 38.

[26]"The Hacker Papers," Psychology Today 14(Aug. 1980), p. 67. Psychologist Philip Zimbardo is credited with discovering "The Hacker Papers" on Stanford's Low Overhead Time-Sharing System (LOTS). Basically, the finding consists of a series of exchanges between a group of undergraduate hackers and ex-hackers at the university. Their main topic of discussion is the pros and cons of hacking from a psychological/sociological standpoint.

[27]Landreth, p. 60.

[28]Ibid., p. 64.

[29]Craig Brod, Technostress: The Human Cost of the Computer Revolution, (Reading, MA, Addison-Wesley Publishing Company, 1984), pp. 16-17.

[30]Ibid., p. 98.

the places which they frequent, in order to maximize interaction with the computer and minimize exposure to all else. In particular, many hackers who are in college begin to practically live in the computer buildings on their respective campuses.[31] This results in a type of "environmental" isolation from the world of non-hackers, which can be a strong source of reinforcement of hacker behavior, since there is often very little to do in a computer building at a college besides work with computers.

In addition, because it is often easiest to work on computer systems undetected during the wee hours of the night and most young hackers attend school by day anyway, they tend to spend many late nights in front of the computer console.[32] Some hackers who are college students even develop a 36-hour cycle: they spend twenty-four hours awake hacking and sleep for twelve hours.[33] For many college hackers, meals often consist of "junk food" since that is the easiest way for them to eat and not have to leave the computer.[34] If they eat elsewhere, they frequent restaurants that are open twenty-four hours a day because of their unusual schedule, or they opt for take-out food. In fact, Chinese food tends to be a favorite among college hackers.[35]

The process of being introduced into the hacker culture continues with changes in their social communication skills. One hacker at Stanford notes that many young hackers are so intelligent that they experienced problems dealing with people long before they discovered hacking.[36] However, the narrow focus of their lives and the irregular hours that they keep only serve to compound their social problems. Every aspect of their lives soon becomes linked to computers. Their speech becomes short and precise in order to avoid any ambiguity in communication, and they often use computer jargon which non-hackers do not understand.[37] Hackers also tend to isolate their emotions from others so that they can maintain control: in effect, they see a relationship with a machine as more predictable, less risky, and more productive than relationships with people.[38] As a result of the combination of the above factors, many young hackers become distant from their families and non-hacker friends.[39] This serves to make them even more reliant on the computer and reinforces their hacker behavior.

However, young hackers are never totally alone: they find refuge among their fellow hackers. In fact, the bond that is created between hackers is probably the biggest reinforcement of hacker behavior. Considering the fact that teenagers tend to be strongly influenced by their peers, this is not surprising. Within this culture, individual achievement is highly recognized. Young hackers love to share information with each other concerning their hacker exploits.[40] It is also described as an extremely understanding culture:

> It is a culture of people who have grown up thinking of themselves as different, apart, and who have a commitment to what one hacker described as "an ethic of total toleration for anything that in the real world would be considered strange." Dress, personal appearance, personal hygiene, when you sleep and when you wake, what you eat, where you live, whom you frequent -- there are no rules. But there is company.[41]

Although the hacker culture may appear tolerant by the standards of the outside world, it does have one very important rule: most young hackers do not look favorably upon those hackers who turn to intentionally doing damage to systems or begin to use their hacking skills for personal or financial gain. As a result, hackers who engage in such activities often find themselves isolated from the hacker culture as well as the world of non-hackers. This occurs because most young hackers see hacking as a serious activity which is not to be taken lightly. Thus, they feel that, when some hackers misuse their skills, they are showing disrespect both for hacking itself and for the other hackers whose work they may be disrupting. In addition, the majority of young hackers do not like the fact that publicity about the antics of abusive hackers ruins the reputation of hackers in general.[42]

CHANGING HACKER BEHAVIOR

Most young hackers are quite defensive of their activities and state that their behavior is neither dangerous to society nor is it harmful to themselves. They often argue that they are providing a valuable service to society because, through hacking, they expose many security problems of which system operators and owners were previously unaware.[43] They justify the large amounts of time that they spend in front of the computer by likening hacking to other creative activities. For example, they state that musicians must spend many hours practicing in order to develop their musical talent to its fullest extent; the same is true of hackers and their talent for working with the computer.[44]

Yet, when young hackers defend their activities with the above arguments, they are overlooking two major points. First, as it was explained in the introduction to this paper, there is a possibility of inadvertent damage to systems and data through hacking and also a potential for disruption by

---

[31] "The Hacker Papers," p. 64.

[32] Ibid., p. 64.

[33] Turkle, p. 232.

[34] "The Hacker Papers," p. 64.

[35] Turkle, pp. 214-215.

[36] "The Hacker Papers," p. 63.

[37] Turkle, p. 202.

[38] Ibid., p. 217.

[39] "The Hacker Papers," p. 63.

[40] Turkle, p. 215.

[41] Ibid., p. 213.

[42] Landreth, p. 68.

[43] "The Constitution of a Hacker," 2600, 1(March 1984), p. 1.

[44] "The Hacker Papers," p. 67.

unfriendly parties who take advantage of unknowing hackers. In addition, hackers really are causing themselves harm: it is unfortunate to see bright individuals become caught in a world that is so narrow-minded in its views. Young hackers have little interest in anything else, and, as a result, they are limiting their opportunity to develop their minds to their fullest capability. It is not enough to lecture them about computer ethics with the hope that it will lead to a change in hacker behavior. Their basic personality characteristics cannot be easily changed,[45] so, in order to promote any effective change in their behavior, the social and environmental forces within the hacker culture which this paper has shown to be quite strong need to be actively addressed.

It appears as if the best way to combat the narrow-mindedness that young hackers often develop is to change educational patterns so that bright students with an interest in computers are not bored in school. If their learning process is made to be more challenging, they will be encouraged to get involved in it. This may lead to them developing interest in other areas in addition to working with computers. Indeed, many activities have "highly-ordered" aspects to them which could attract the young hackers' attention in connection with the part of their personalities which loves structure. For example, one psychologist notes the "antisensual" nature of the interests in the arts and literature of a select group of young hackers which she observed. In particular, they liked music which emphasizes intricacy of structure instead of emotion, such as Bach's many preludes and fugues. Their taste in literature leaned toward science fiction, mainly because such writings emphasize the structure of worlds rather than the definition of character. In art, they expressed a fondness for Escher prints. These tend to contain recursive elements, such as a picture of hands drawing each other (see Figure 2 below).[46]

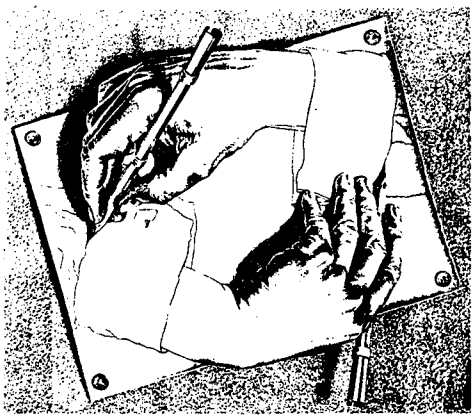


Figure 2: Escher's Drawing Hands

Reproduced from Escher's The Graphic Work of M.C. Escher, (New York, Ballantine Books, 1971), p. 69.

[45]This tenet has long been accepted by the world of psychology. It has been espoused by many leaders in the field, including Sigmund Freud and William James. However, the subject was recently reopened for debate due to the impact of some new research. For further information, see Zick Rubin, "Does Personality Really Change After 20," Psychology Today, 15(May 1981), pp. 18-27.

[46]Turkle, pp. 219-222.

Research has shown that a preference for such "antisensual" aspects of the arts and literature, especially science fiction, exists among students who are interested in "investigative" occupations outside the computer science field, such as medicine, engineering, and scientific research.[47] This commonality of interests suggests that increased interaction with peers who are not involved in hacking is a distinct possibility for young hackers.

Thus, the purpose of developing young hackers' interests in activities outside computers is threefold: it would further develop their intellectual abilities, give them exposure to different environments, and enhance their social skills. In the end, these three points could lead to a decrease in hacker behavior. The development of special programs for gifted students which feature a balance of challenging academic work and well-planned extracurricular activities, such as Duke University's Talent Identification Program (TIP),[48] is a step in the right direction.

The communications skills of young hackers could also be helped by education in two other forms. By providing their teachers, peers, parents, and law enforcement officials with more and better basic information about the workings of computers, young hackers and these other groups would be better able to understand one another. Also, by allowing young hackers to work with computers in an environment where the point of their work is more focused and supervision is available, they would be able to continue to enjoy working with computers at the same time that they are experiencing them as a tool with a purpose beyond the realm of hacking. Cooperative education or work study programs would be excellent strategies for doing this.

CONCLUSION

The analysis of young hackers' behavior which this paper has presented is a complex one: hackers have basic personality characteristics which attract them to work with computers and which later play a role in their initiation into the hacker culture. Strong social and environmental influences within the hacker culture also interact with their personality traits to reinforce hacker behavior. As a result, helping young hackers to change their behavior is not an easy proposal. The methods that have been suggested in this paper involve extensive planning, commitment, patience, and, of course, financial backing. Yet, given the analysis of their behavior which has been provided, there is an excellent chance that the plans for working with hackers that are listed would be successful if they were properly implemented. In the end, the benefits for both society and young hackers themselves could be well worth the effort.

[47]Julie A. Smith, "An Observation of the Career Decision-Making Process of Honors College Students," unpublished Thesis submitted to Michigan State University, E. Lansing, MI, in partial fulfillment of the requirements for the Honors degree of Bachelor of Arts, Department of Psychology, 1982.

[48]For information on this and similar programs, see John Boslough, "Challenging the Brightest," Psychology Today, 18(June 1984), pp. 28-33.

REFERENCES

Boslough, John, "Challenging the Brightest," Psychology Today, 18, June 1984, pp. 28-33.

Brod, Craig, Technostress: The Human Cost of the Computer Revolution, Reading, MA, Addison-Wesley Publishing Company, 1984.

Campbell, Patricia F., and McCabe, George P., "Predicting the Success of Freshmen in a Computer Science Major," Communications of the ACM, 27, Nov. 1984, pp. 1108-1113.

"The Constitution of a Hacker," 2600, 1, March 1984.

"Death Star Cards Spell Woe," 2600, 1, Feb. 1984.

Deaux, Kay, The Behavior of Men and Women, Monterey, CA, Brooks/Cole Publishing Company, 1976.

Escher, M.C., The Graphic Work of M.C. Escher, New York, Ballantine Books, 1971.

Gorney, Cynthia, "Hack Attack: Computer Whiz Kids Harass Magazine Reporter," The Washington Post, Dec. 6, 1984, p. B1.

"The Hacker Papers," Psychology Today, 14, Aug. 1980, pp. 62-69.

Kiesler, Sara, Sproull, Les, and Eccles, Jacquelynne, "Second-Class Citizens," Psychology Today, 17, March 1983, pp. 40-48.

Landreth, Bill, Out of the Inner Circle: A Hacker's Guide to Computer Security, Bellevue, WA, Microsoft Press, 1985.

Levy, Steven, Hackers: Heroes of the Computer Revolution, Garden City, NY, Anchor Press/Doubleday, 1984.

Perry, Tekla S., and Wallich, Paul, "Can Computer Crime Be Stopped," IEEE Spectrum, 21, May 1984, pp. 34-45.

Rubin, Zick, "Does Personality Really Change After 20," Psychology Today, 15, May 1981, pp. 18-27.

Sandza, Richard, "The Night of the Hackers," Newsweek, 104, Nov. 12, 1984, p. 17.

Sandza, Richard, "The Revenge of the Hackers," Newsweek, 104, Dec. 10, 1984, p. 17.

Shea, Tom, "The FBI Goes After Hackers," Infoworld, 6, March 26, 1984, p. 38.

Smith, Julie A., "An Observation of the Career Decision-Making Process of Honors College Students," unpublished Thesis submitted to Michigan State University, East Lansing, MI, in partial fulfillment of the requirements for the Honors degree of Bachelor of Arts, Department of Psychology, 1982.

Strong-Campbell Interest Inventory, Form T325, distributed by National Computer Systems, Inc., Minneapolis, MN, 1981.

Turkle, Sherry, The Second Self: Computers and the Human Spirit, New York, Simon and Schuster, 1984.

"2600 Writer Indicted," 2600, 1, June 1984, p. 3.

U.S. Department of Labor, Bureau of Labor Statistics, Occupational Outlook Handbook, Washington, D.C., April 1984.

Weizenbaum, Joseph, Computer Power and Human Reason: From Judgment to Calculation, San Francisco, CA, W.H. Freeman and Company, 1976.

Williams, Dennis A., and Sandza, Richard, "Teaching Hackers Ethics," Newsweek, 104, Jan. 14, 1985, p. 76.

MULTILEVEL SECURITY FROM A PRACTICAL POINT OF VIEW

Terry S. Arnold

Merdan Group, Inc.
4617 Ruffner St.
San Diego, CA 92111
619/571-8565

ABSTRACT

The technology base for multilevel secure computer systems has been evolving over the past 15 years. With appropriate development constraints this technology is sufficiently mature to be incorporated in the current generation of new C3I systems. By system we mean both a standalone system and a network of "systems." This paper addresses these constraints from the perspectives of concept formulation and actual development. The process of defining these constraints and the pitfalls which must be avoided are described. The management posture needed for successful multilevel secure development is presented.

INTRODUCTION

The multilevel security (MLS) issue has been widely discussed over the past 15 years. The positions taken by various people range from that it is impossible to "nothing less than perfection is acceptable." This paper presents the views of one practitioner who believes that it is currently feasible to implement MLS, as long as appropriate constraints are applied by management. The emphasis of this paper is to define the impact of multilevel security on the C3I development process and in particular the management issues involved.

WHY MULTILEVEL

Why we need multilevel secure operation is a question that many people ask. The reasons are very basic and near and dear to the C3I manager's heart. The basic reason is that C3I is inherently multilevel due to the fact that compartmentation is required for some of the I data. When one thinks of applying a system high policy where many compartments are involved it becomes clear that this type of policy does not make sense. In addition to this aspect, successful implementation of multilevel security will allow cost effective sharing of expensive computer resources. One of the biggest benefits lies in that multilevel secure operation will allow controlled information sharing within the C3I community.

WHAT DOES MULTILEVEL SECURE MEAN?

For a system to be multilevel secure means several things. The first and most significant is that we trust a computer to enforce our security policy with respect to all of our data. The means by which this security policy is enforced has several aspects. The primary methods are to separate data based on differing levels of classification/compartmentation and strictly control user access. These concepts are not new to the world of procedural security. The only thing that is new is that a computer is used as a surrogate System Security Officer (SSO). One of the functions of this automated SSO is to make a log of all attempted or apparently attempted security violations.

NEEDED TECHNOLOGY

The technology needed to support multilevel security covers most of the computer science spectrum. First and foremost is the concept of the reference monitor. The reference monitor is the automated embodiment of the SSO. We need a rigorous expression of our security policy in the form of a security model. We need methods for verifying that our security policy is in fact being enforced by the implementation. Lastly we need computer architectures which will efficiently support the reference monitor.

CURRENT TECHNOLOGY

The current state of the art in multilevel security is evolving at a fairly rapid rate. While there are still some holes in the technology base, research is well under way to fill in the gaps. We have abstract mathematical models which have been shown adequate to describe most aspects of Department of Defense (DoD) security policy. The Bell-LaPadula model developed at MITRE is the most widely accepted such model. Practical application of this model quickly revealed that real systems need some exceptions to this model. While some people prefer to wave hands in this area, progress is being made in that concrete models are being put together for real systems. The fact that concrete models of what it means for a given system to be secure (i.e., a rigorous statement of the security policy) are being constructed bodes well for application of multilevel security technology in the C3I community. At the present time such concrete modeling is not wide spread even in the computer security community but with time and

applied determination we will carry the day. The situation in the area of verification methods is somewhat less rosy. We do have methods for formally specifying and verifying multilevel security at a fairly high level of abstraction. Problems arise in two areas. The first is in the area of exceptions to the Bell-La Padula model, where some of the methods do not have a means for expressing the allowed exceptions within their notation. Several research groups are actively working to eliminate this difficulty. We expect positive results in the near future. The second area where problems arise is verification of the actual implementation. Automated verification of software has been a research topic for a number of years. At the present time we do not have viable automated tools to support verification of software implementations. Some research groups are working in the area but solid results may be several years off. At the present time we must use manual methods which are labor intensive. Unfortunately the labor resource needed (security trained software engineers) is in short supply and tend to "burn out" on this type of work. On the brighter side, security kernel designs embodying the reference monitor concept are starting to appear. Several have actually been implemented and certified to operate in the multilevel secure mode. Securable computer architectures are becoming common, with securable microprocessors starting to be produced in production quantities.

EXPERIENCE TO DATE

The experience with implementing multilevel security has met with mixed success, although even the failures have added greatly to the experience base of the computer security community. The initial effort to use MULTICS as a base for a multilevel secure operating system for the Air Force Data Services Center produced what has to be considered a classic penetration study. The problems identified were remedied through what we believe to be the first practical operational application of modern multilevel security technology. The SACDIN system fostered the development of much of the technology base that we have to draw upon. At this point in time it is not yet operational but the prospects are excellent. The AN/GSC-40 was an effort to implement multilevel security for a special purpose network control application. It is operational today and represents to our knowledge the first successful application of modern multilevel security technology in an operational environment. The AUTODIN II project attempted to apply the then state of the art of multilevel security to building a replacement for AUTODIN. The project demonstrated that formal specification and verification are practical for a large scale system, however it became clear that the software development process must be tightly controlled. The KVM-370 project attempted to apply the reference monitor concept to an existing commercial operating system. The project appears to have been successful but there are reports that performance is less than optimum. The SCOMP project is a successful commercial effort to produce a multilevel secure operating system. This effort is particularly notable, since it was submitted to the DoD Computer Security Center for certification at the A1 (i.e., highest) level. The certification effort was successful and SCOMP was certified at the A1 level December 24, 1984. The success of this effort, albeit at the cost of considerable effort and performance penalties, provides strong evidence that the prospect for off the shelf multilevel secure operating systems is improving. The projects described above are only part of the experience base of the computer security community but they are all in their own way landmarks in the evolution of this technology. A number of new programs are underway to incorporate multilevel security technology in real world systems. In particular I-S/A AMPE, Regency Net, and BLACKER are rather serious about achieving multilevel security as part of their project goals.

LESSONS LEARNED

As a result of the efforts described above a great deal has been learned about what it really takes to achieve multilevel security. The first and possibly most important lesson is perhaps typified by the WW II expression "Keep it simple stupid." The attempts at generality have either resulted in failure or poor performance. Security models need to be tailored for the application, since the general models do not address the specifics of the real world. Formal (in some sense) specifications of what a given system is supposed to do correctly are needed. In the absence of such specifications we have difficulty in determining that we have a secure system. Securable computers are becoming very common, since the architectural features that are needed for general applications are similar to those required to support a reference monitor. Painful experience has taught us that standard software engineering practice is not good enough to provide the quality of software needed for multilevel security. This not a failing of software engineering technology but of management of the software development process. Verification of the product of the software engineering process is needed and must occur in parallel with it.

TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA

The DoD Computer Security Center (DODCSC) was established in 1981. The mission of the DODCSC is to serve as a focal point for computer security throughout the DoD. One of its major accomplishments has been the publication in 1983 of the Trusted Computer System Evaluation Criteria (TCSEC). This landmark document defines eight evaluation categories for trusted computer systems. While the main thrust of the TCSEC is directed toward mainframe general purpose systems, they are being successfully applied to systems which employ embedded computers. The TCSEC define fairly specific criteria which a computer system must meet to be evaluated at a given level. The one criticism which has been made is that the TCSEC do not define the category that a given system must fall into for it to be considered adequately secure. While some may view this as a fatal flaw, it has not hampered application of the criteria in that the serious workers in the field are in agreement about what levels are appropriate for a given application. Environment criteria which eliminate the above criticism will very likely have been issued by the time that this paper is presented. Very recently a effort to develop criteria for networks was started. The progress to date is significant and the prospects for having workable criteria in the near future are excellent.

The following evaluation categories are defined in the TCSEC:

Beyond A1

A1 Verified Design

B3 Security Domains

B2 Structured Protection

B1 Labeled Security Protection

C2 Controlled Access Protection

C1 Discretionary Security Protection

D Minimal Protection

These criteria call out increasing levels of required security features and development requirements as one proceeds from category D upward on the scale. Determination of what level a given system actually meets is one of the elements of the charter of the DODCSC. Unfortunately only a small number of commercially available systems have been completely evaluated, with respect to the Criteria, at this time. This state of affairs is going to change in the near future as more and more vendors are submitting products for evaluation. The impact that this will have on the C3I community will increase with time, in that some of the major computer vendors are actively working the problem. However, for new program starts in the next year or so the C3I PM will have to "roll his own" security design and start from scratch in the evaluation area. While this may seem to be a somewhat sad state of affairs it should not come as much of a surprise, since many of the C3I systems in place or under development use custom operating systems rather than computer vendor supplied "off the shelf" operating systems. The DoD efforts toward standardization of hardware and software should improve this situation significantly over the next 5 years. Even when these products finally are available a C3I PM is still going to have to apply good security engineering practice to build his system on these foundations.

HOW TO DO IT RIGHT

All of this technology would go to waste unless there was a systematic approach to implementing multilevel security for a given application. Such approaches have evolved over time and can be summarized by the following four seemingly simple steps.

DEFINE IT SECURE

BUILD IT SECURE

PROVE IT SECURE

KEEP IT SECURE

The first step is where security models come into the picture. Defining up front exactly what it means for a given system to be secure is a very important first step. It is a very good idea to choose a TCSEC evaluation category at this point in time. The second step is really the toughest one in that the temptations to cut corners during the development process abound. Most software developers will strongly object to the constraints which must be placed on them, in order to successfully perform this step. We will address these constraints in detail below. The third step will be successful only if the second step was done properly. The term "PROVE" has many interpretations in the computer security community. These interpretations range from the somewhat naive concept of pure testing to the extreme of mathematical proof of correctness. The TCSEC play a major role in that they define levels of checking that are appropriate. The last step is the simplest in that it consists of little more than configuration management coupled with procedural security.

DEVELOPMENT CONSTRAINTS

As we mentioned earlier constraints need to be placed on the development process

if multilevel security is to be successfully implemented. Most of these constraints are derived from common sense but the necessity of them has been learned the hard way. The first constraint is to perform all development in a secure environment. The importance of this has only very recently become public with the recent rash of "hacker" break-ins to what many people thought were "secure" computer systems. The usual means for providing a secure environment is to use a dedicated/closed computer for all development and to treat all software as if it were classified. The second constraint is to recognize that security must drive the design. This may cause some difficulty in that most software developers will want to reuse previously developed programs which were not designed or developed with security as a driving requirement. A security model of the application should be the first order of business. This model should be a concrete statement of what it means for the application to be secure. A formal specification of how the security model will be enforced is the next item which needs definition. There is a strong tendency for the technical types to start waving their hands at this stage, since few of them understand the role of a formal specification. To most of them MIL-STD-490 is a millstone around their neck to which they pay only lip service. The degree of formality for the specification may vary but the emerging techniques which have a strong theoretic basis are far and away the best. When a mathematically based formal specification method has been used it is feasible to show that the specification satisfies the security model and thus describes a secure system. Once the specification has been shown secure then it is appropriate to start detail design. There is a very strong tendency to start the detail design before the specification is shown to be secure. This may seem like a time saver but experience has shown that this is not the case. Verification of the detail design is a necessary step in that most if not all designs initially exhibit significant security flaws. Verification of the actual implementation should be performed, since testing has the unenviable record of showing only the presence of errors and not their absence. In one case testing lulled a program manager into believing that a security flaw exposed by verification in fact did not exist. This PM was rather shocked when the "test" suddenly displayed the flaw some months later after a few "minor" software changes.

MANAGEMENT ISSUES

In the material presented above it has become clear that significant management issues arise when multilevel security is involved. The first issue is whether a computer is really needed. Often there are hardware solutions which can make the job easier. Hardware solutions are preferable, since they are better understood and more easily analyzed. The second issue is what does it mean for your system to be secure. This is clearly important, since it will almost always have a major impact on the overall design. The third issue is what are your accreditation/certification requirements. Each of the services has some form of accreditation/certification regulation. A key step is to determine what your specific requirements are in this area. Getting the accreditation/certification authority in at the start is a critical step. Six months before IOC is a bit late if success is a desired objective. Require the developing contractor/agency to provide all of the data needed for security evaluation. This seems like a minor point but the standard data items don't provide sufficient technical data for security evaluation. Lastly stick by the rules come hell or high water.

CONCLUSIONS

Multilevel security is feasible today when appropriate constraints and technology are employed. The needed technology exists in that we know what has to done and how to do it right. We have a gap in that we cannot just take products off the shelf and use them to solve our security problems. Management by the book is required for success. Unfortunately only part of the "book" exists today. Efforts are underway to flesh out the "book" particularly in the area of data item descriptions. The light at the end of the tunnel is getting brighter and it is looking a lot less like a train coming the other way.

MODELING OF COMPUTER NETWORKS

Dr. R. A. Gove
Booz, Allen and Hamilton, Inc.
4330 East West Highway
Bethesda, MD 20814
(301) 951-4624

INTRODUCTION

In this paper we will discuss two formal techniques for modeling computer networks. The first approach is an access control approach based on the Bell & LaPadula (BLP) model scheme as given in [1]. The second approach models the network in terms of moving messages between hosts. Table 1 provides a summary of the features of the BLP model.

In the BLP model, each state $v \in V$ has three components. These components will be denoted by access[v], current_level[v], and matrix[v]. Recall that:

- access[v] $\subset S \times O \times A$ and represents the access attributes a subject has with respect to an object in state v.
- current_level[v]: $S \rightarrow L$ gives the current security level of each subject in state v.
- Matrix[v](s, o) $\subset A$ is the set of attributes that s is allowed with respect to o.

Recall that a state $v \in V$ is said to satisfy the simple security property if and only if the following condition holds:

> For all $s \in S$ and for all $o \in O$,
> $(s, o, \underline{r}) \in$ access[v] implies
> current_level[v](s) $\geq f_{obj}(o)$.

In words, this says that if a subject has read access to an object, then the clearance of the subject dominates the classification of the object. Also recall that a state $v \in V$ satisfies the *-property if and only if the following conditions hold:

> For all $s \in S$ and for all $o \in O$
> $(s, o, \underline{a}) \in$ access[v] implies
> $f_{obj}(o) \geq$ current_level[v](s)
>
> $(s, o, \underline{w}) \in$ access[v] implies
> $f_{obj}(o) =$ current_level[v](s)

Table 1

A Bell & LaPadula computer security model is an 11-tuple
$B = (S, O, V, R, \rho, A, L, \geq, f_{sub}, f_{obj}, v^*)$

Symbol	Description	
S	Subjects:	Set of active entities
O	Objects:	Set of passive entities
V	States:	States of system accessess
R	Requests:	Requests for access modes
ρ	Rho:	State changing function; rules
A	Attributes:	r = read only, w = read/write, a = append, c = control, e = execute
L	Levels:	Set of security levels
$\geq$	Is dominated by:	Order relation on L
f_{sub}	Subject level:	Clearance level of members of S
f_{obj}	Object level:	Classification of members of O
v^*	Initial state:	The starting state of the system

$(s, o, \underline{r}) \in access[v]$ implies
$f_{obj}(o) \leq current\_level[v](s)$

A system state that satisfies both the simple security property and the *-property is called a secure state. It was shown in [1] that if the initial state v* is secure and if ρ preserves the simple security and *-properties, then all states are secure and, hence, the system is secure. In the next section we will show how to apply the BLP model schema to a network of computers.

BELL & LaPADULA NETWORKS

Let $B_1, B_2, \ldots, B_n$ be BLP models of a network of n host computers.

Let $B_i = (S_i, O_i, V_i, R_i, \rho_i, A, L, \geq, (f_{sub})i, (f_{obj})i, v^*_i)$.

Note that we are assuming that all the B_i have the same attribute set and the same dominance relation and set of levels. Associated with each host are the maximum and minimum security levels at which the host operates. These levels are denoted by $max(B_i)$ and $min(B_i)$, respectively.

The first step in defining the network model is to define the subjects. Consider the subject sets S_i of the individual hosts. Since many hosts will have subjects in common, $S_i \cap S_j$, may be non-empty. We will distinguish these common elements with a construct known as the disjoint union. Technically, the disjoint union of A and B is

$$A \amalg B = A \times \{0\} \cup B \times \{1\}.$$

Generally, A is identified with $A \times \{0\}$ and B with $B \times \{1\}$ so that $A \cap B = 0$. The construction extends to any finite list of sets in the obvious way.

Let S_{Net} be the disjoint union of the individual S_i:

$$S_{Net} = \coprod_{i=1}^{n} S_i$$

The network's attribute set will include the standard attributes $\underline{r}$, $\underline{w}$, $\underline{a}$, $\underline{e}$, and $\underline{c}$ of the individual models but will also have two additional attributes: $\underline{k}$ and $\underline{l}$. The attribute $\underline{l}$ represents "logon." A subject receiving log on access to a host computer is given access to the host's resources. (This implies, of course, that the host must be considered as an object.) The attribute $\underline{k}$ represents connect access. A subject in host i may request to be connected through the network to host j. As we will show below, an $\underline{l}$ request has to be preceeded by a $\underline{k}$ request. We have $A_{Net} = \{\underline{r}, \underline{e}, \underline{a}, \underline{e}, \underline{c}, \underline{l}, \underline{k}\}$.

The objects of the network are formed from the individual objects of each host with the addition of the hosts themselves. Let O_i be the objects for B_i. Then

$$O_{Net} = \coprod_{i=1}^{n} O_i \amalg \{B_1, \ldots, B_n\}.$$

The subject and object clearance functions are easily described.

$$f_{sub}(s) = (f_{sub})_i(s) \quad \text{if } s \in S_i \text{ for some } i.$$

And

$$f_{obj}(o) = \begin{cases} (f_{obj})_i(o) & \text{if } o \in O_i \text{ for some } i \\ max(o) & \text{if } o = B_i \text{ for some } i \end{cases}$$

The states of the network are formed just as in the BLP host model. That is, a network state v is a 3-tuple,

$(access_{Net}[v], current\_level_{Net}[v], matrix_{Net}[v])$.

where

$access_{Net}[v] \subset S_{Net} \times O_{Net} \times A_{Net}$

$current\_level_{Net}[v]: S_{Net} \rightarrow L$

$matrix_{Net}[v]: S_{Net} \times O_{Net} \subset subsets(A_{Net})$.

The initial state, v*, is formed from the initial states

$v_i^* \in V_i^*$, $i = 1, 2, \ldots, n$.

Let
$v_i^* =$
$(access_i[v_i^*], current\_level_i[v_i^*],$
$matrix_i[v_i^*])$.

Define

$$access_{Net}[v^*] = \coprod_{i=1}^{n} access_i[v_i^*]$$

$$current\_level_{Net}[v^*] = \coprod_{i=1}^{n} current\_level_i[v_i^*]$$

$$matrix_{Net}[v^*] = \coprod_{i=1}^{n} matrix_i[v_i^*]$$

Some special remarks about the initial $matrix_{Net}$ are needed here. As part of defining the host's relationship to the net, we will consider the initial BLP model of each host to have included in its initial matrix the appropriate $\underline{k}$ and $\underline{l}$ accesses. That is, for each B_j, $j \neq i$, to which a subject $s \in S_i$ is allowed to connect, $\underline{k} \in matrix_i[v_i^*](s, B_j)$. Note that the connect accesses are controlled by the source host, and will be monitored by its reference monitor. The $\underline{l}$ accesses are treated somewhat differently. If $s \in S_i$ and s has log on access to B_j, then $\underline{l} \in matrix_j[v_j^*](s_j, B_j)$. Here, s_j is the copy of s that is in the subject set for B_j. Thus, the target host maintains the log on access list for all of a subject's alter egos.

The simple security property and the *-property are defined exactly as they are in the host model. A state in V_{Net} is secure if and only if it satifies both simple security and *-property and current $level_{Net}[v]$ is consistent. That is, if $s \in S_i \subset S_{Net}$ and s_j is the copy of s in S_j, then $current\_level_{Net}[v](s) = current\_level_{Net}[v](s_j)$. The net is secure if all the states are secure. We first prove

Proposition 1

If v satisfies the *-property and $s \in S_{Net}$, $o \in O_i$ and $o' \in O_j$ and $(s, o, \underline{r}) \in access_{Net}[v]$ and $(s_j, o, \underline{w}) \in access_{Net}[v]$ then $f_{obj}(o') \geq f_{obj}(o)$.

Proof. The *-property implies that $current\_level_{Net}[v](s) \geq f_{obj}(o)$ and $current\_level_{Net}[v](s_j) = f_{obj}(o')$. Consistency implies that $f_{obj}(o') \geq f_{obj}(o)$.□

We now prove

Proposition 2

If, for all i=1,..., n, v_i^* is a secure state in B_i, then v^* is a secure state.

Proof. We first prove the simple security property. Suppose $s \in S_{Net}$, $o \in O_{Net}$ and $(s, o, \underline{r}) \in access_{Net}[v^*]$. From the definition of $access_{Net}[v^*]$, there is an i with $(s, o, \underline{r}) \in access_i[v_i^*] \subset S_i \times O_i \times A$.
Thus, $s \in S_i$ and $o \in O_i$.
Since v_i^* is secure,
$current\_level_i[v_i^*](s) \geq (f_{obj})_i(o)$.
But for $s \in S_i$,
$current\_level_{Net}[v_i^*](s) = current\_level_i[v_i^*](s)$, and so the simple security condition holds in the net.

To prove that the *-property holds for v^*, consider $(s, o, \underline{a}) \in access_{Net}[v^*]$. From the definition of $access_{Net}$, we conclude that $(s, o, \underline{a}) \in access_i[v_i^*]$ for some i. Since v_i is a secure state, $(f_{obj})_i(o) \geq current\_level_1[v_i](s)$.
But $f_{obj}(o) = (f_{obj})_i(o)$ for $o \in O_i$ and $current\_level_{Net}[v^*](s) = current\_level_i[v_i^*](s)$ for $s \in S_i$.
The proofs of the remaining two conditions for *-property are similar and are left to the reader.□

We next must define state transitions and show that the state transitions preserve simple security and *-properties. The transition function will be built up from the individual ρ_i plus some additional commands: Connect_to_host and logon to_host. These two new commands are of the form:

s requests connect_to_host_access_to_B_j at_level_ λ.

s requests logon_to_host_access_to_B_j_at level_ λ.

We will also allow the complement of these comments; that is, delete log on access (log off) and delete connect access.

Let R* be the set of requests generated by this schema. Define:

$$R_{Net} = \coprod_{i=1}^{n} R_i \amalg R^*$$

We have defined the states for the network and the set of requests. In order to complete the definition of the BLP network model, it is necessary to define the state change function:

$$\rho_{Net}: V_{Net} \times S_{Net} \times R_{Net} \rightarrow V_{Net}$$

Suppose $v \in V_{Net}$, $s \in S_{Net}$ and $r \in R_{Net}$. There exists some i, i=1,...,n, with $s \in S_i$. Likewise, $r \in R_j$ for some j or else $r \in R^*$. The state v is characterized by

$$access_{Net}[v] \subset S_{Net} \times O_{Net} \times A_{Net}$$

$$current\_level_{Net}[v]: S_{Net} \rightarrow L$$
$$matrix_{Net}[v]: S_{Net} \times O_{Net} \rightarrow Subset(A_{Net}).$$

There are several cases to consider: whether the request r is in R_i, R_j, or R*. Although the constructions below look rather complicated, all we are really doing is letting ρ_i change the state by acting on the ith model.

Case 1

$r \in R_i$. (In this case, s is requesting an access in its own host).

Let b_i = $access_{Net}[v] \cap (S_i \times O_i \times A)$.

That is, b_i is the subset of accesses that relate to host i. Similarly, let f_i be $current\_level_{Net}[v]$ restricted to the subdomain S_i,

$$f_i = current\_level_{Net}[v] | S_i.$$

And similarly for the matrix,

$$m_i = matrix_{Net}[v] | S_i \times O_i.$$

It follows that (b_i, f_i, m_i) defines a state $v_i \in V_i$ of the host B_i. We then let ρ_i act on v_i obtaining $\tilde{v}_i = \rho_i(v_i, s, r)$. We then form $\tilde{v}$ by replacing b_i, f_i, and m_i with the corresponding components of $\tilde{v}_i$:

$$access_{Net}[\tilde{v}] = (access_{Net}[v] \sim b_i) \cup access_i[\tilde{v}_i]$$

$$current\_level_{Net}[\tilde{v}](t) = \begin{cases} current\_level_{Net}[v](t), & t \notin S_i \\ current\_level_i[\tilde{v}_i](t), & t \in S_i \end{cases}$$

$$matrix_{Net}[v](t,o) = \begin{cases} matrix_{Net}[v](t,o), & t \notin S_i \text{ or } o \notin O_i \\ matrix_i[\tilde{v}_i](t,o), & t \in S_i \text{ and } o \in O_i \end{cases}$$

Case 2

$r \in R_i$. (In this case, s is requesting an access in another host). The philosophy behind the action of ρ_{Net} in this case is to let the target host use its reference monitor to control the request of subject s. This is done by transferring requests by $s \in S_i$ to requests by a copy s_j of s in S_j. The subject s must first obtain connect access and log on access to the host B_j. That is, the following condition is checked:

$$(s_j, B_j, \underline{1}) \in access_{Net}[v].$$

If this condition holds, the next state is determined by ρ_j much as was done in case 1. That is, let

$b_j = access_{Net}[v] \cap (S_i \times O_i \times A)$

$f_j = current\_level_{Net}[v] \mid S_j$

$m_j = matrix_{Net}[v] \mid S_j \times O_j$.

Then (b_j, f_j, m_j) is a state $v_j \in V_j$ of host B_j. We let $\tilde{v}_j = \rho_j(v_j, s_j, r)$ where s_j is the copy of s in S_j. Then v is defined by

$access_{Net}[v] = (access_{Net}[v] \sim b_j) \cup access_j[v_j]$

$current\_level_{Net}[\tilde{v}](t) =$
$current\_level_j[\tilde{v}_j](t)$, $t \in S_j$
$current\_level_{Net}[vj](t)$, $t \notin S_j$

$matrix_{Net}[v](t,o) =$
$matrix_{Net}[v](t,o)$, $t \notin S_j$ or $o \notin O_j$
$matrix_j[\tilde{v}](t,o)$, $t \in S_j$ and $o \in O_j$

Case 3

$r \in R^*$. (In this case, s is requesting either a connect access or a log on access)

Subcase 3a. r = "request_connect_to_host access_to_B_j_at_level_ λ."

In this case, the following conditions are checked.

(i) $\underline{r} \in matrix_{Net}[v](s, B_j)$

(ii) $current\_level_{Net}[v](s) = \lambda$

(iii) $min(B_j) \leq \lambda \leq max(B_j)$

If any of these conditions do not hold, the new state is the old state v. If these conditions are satisfied, then the new state v is defined by adding connect access, $\underline{k}$, to the network access set. This is done, however, in the disjoint component corresponding to the target host B_j. Note that the reference monitor of B_i decides if s has discretionary connect access. If it is allowed, then the target host will add connect access to S_j's access, where S_j is the copy of s in S_j as before. The host will also reset S_j's current clearance level to λ. Specifically:

$access_{Net}[\tilde{v}] = access_{Net}[v] \cup \{(s_j, B_j, \underline{k})\}$

$current\_level_{Net}[\tilde{v}](t) =$
$\begin{cases} current\_level_{Net}[v](t), & t \neq s_j \\ \lambda & t = s_j \end{cases}$
$matrix_{Net}[\tilde{v}] = matrix_{Net}[v]$.

Sub-Case 3b. r = request logon_to_host access_to_B_j_at_level_ λ.

In this case, the following conditions are checked:

(i) $(s_j, B_j, \underline{k}) \in access_{Net}[v]$

(ii) $\underline{l} \in matrix_{Net}[v](s_j, B_j)$

(iii) $current\_level_{Net}[v](s_j) = \lambda = current\_level_{Net}[v](s)$

If these conditions are satisfied, the new state v is defined by adding two new accesses to B_j. First $\underline{l}$ access is added to the access set. We also have to make a special adjustment to prevent *-property violations between hosts. That is, if a subject has read access to a Top Secret object, for example, in host B_i and then logs on to B_j (at the Top Secret level) and obtains write access to an object at less that the TS level, we would have a covert channel and the potential for compromise. We avoid this by adding a read access mode for the host B_i as an object in host B_j. This will enable the *-property checks in B_j to catch the potential compromise. Specifically, if there exists an $o \in O_i$ such that $(s_i, o, \underline{x}) \in access_i[v_i]$ for $\underline{x} \in \{\underline{a}, \underline{w}, \underline{r}\}$

$access_{Net}[v] = access_{Net}[v] \cup \{(s_j, B_j, \underline{l})\} \cup \{(s_j, B_i, \underline{x})\}$

Otherwise,

$access_{Net}[\tilde{v}]$ =
$access_{Net}[v] \cup \{(s_j, B_j, \underline{1})\}$.

The current_level and matrix components remain unchanged.

We have now completed the definition of $\rho_{Net}: V_{Net} \times S_{Net} \times R_{Net} \rightarrow V_{Net}$. The components of the network model are summarized in Table 2. We have already defined the network's intitial state and have shown that it is secure if the host's initial states are secure. It remains to show that the state transition mapping ρ_{Net} preserves the simple security and *-properties.

Theorem

If $B_1, \ldots, B_n$ are secure Bell & LaPadula systems and B_{Net} is defined as above, then B_{Net} is a secure Bell & LaPadula system.

Proof. It suffices to show that ρ_{Net} preserves the simple security and *-properties. Let $v \in V_{Net}$ be a secure state, $s \in S_{Net}$ and $r \in R_{Net}$. Let $v = \rho_{Net}(v, s, r)$. Without loss of generality, we may assume $s \in S_1$.

Case 1

$r \in R_1$. In this case,

$access_{Net}[\tilde{v}]$ =
$(access_{Net}[v] \sim b_1) \cup access_{Net}[\tilde{v}_1]$,

where b_1, and $\tilde{v}_1$ are defined as above. Now suppose $t \in S_{Net}$, $o \in O_{Net}$ and $(t, o, \underline{r}) \in access_{Net}[\tilde{v}]$. Suppose $(t, o, \underline{r}) \in access_{Net}[\tilde{v}](t) \sim b_1$. Then $t \neq s$, and $current\_level_{Net}[\tilde{v}](t) = current\_level_{Net}[v](t)$.

Since simple security holds in v, $current\_level_{Net}[v](t) \geq f_{obj}(o)$.

Next, consider $(t, o, \underline{r}) \in access_1[\tilde{v}_1] \subset S_1 \times O_1 \times A$.

Table 2

Bell & LaPadula Network Model

Symbol	Description
B_{Net}	BLP Model for network
S_{Net}	Net Subjects: The disjoint union of the component subjects
O_{Net}	Net Objects: The disjoint union of the component objects plus the hosts
R_{Net}	Net Requests: The disjoint union of the component requests plus log on and connect
ρ_{Net}	The Net Transition function: acts like ρ_i on host i, plus adds connect and log on access
A_{Net}	Net attributes: Host attribute plus $\underline{1}$, $\underline{k}$
L	Classification levels
$\geq$	Dominance relation for levels
f_{sub}	The clearance function for the net subjects. It is the disjoint union of the host clearance
f_{obj}	The object clearance function. It is the disjoint union of the host object clearances
v*	The network initial state. The disjoint union of the initial states of the component hosts.

In this case $current\_level_{Net}[\tilde{v}](t) = current\_level_1[\tilde{v}_1](t)$. Since ρ_1 preserves simple security by assumption, $current\_level_1[\tilde{v}_1](t) \geq (f_{obj})_1(o) = f_{obj}(o)$.

We next show that the *-property holds. Consider $(t, o, \underline{a}) \in access_{Net}[\tilde{v}]$. If $(t, o, \underline{a}) \notin b_1$, then
$f_{obj}(o) \geq current\_level_{Net}[v](t)$
because v is a secure state.
If $(t, o, \underline{a}) \in access_1[\tilde{v}_1]$
$S_1 \underline{x} O_1 \times A$, $current\_level_{Net}[\tilde{v}](t) = current\_level_1[\tilde{v}_1](t)$.

Since ρ_1 preserves *-property,
$f_{obj}(o) = (f_{obj})_1(o) \geq current\_level_1[\tilde{v}_1](t)$.

The proofs of the other two conditions are similar.

Case 2

$r \in R_j$ $1 = j$. The proof of this case is similar to Case 1 and is left to the reader.

Case 3

Since requests in R* do not affect r, a, or w access, simple security and *-properties must be preserved.□

We have defined a network model based on individual BLP models and shown it is secure. In the next section we consider an alternative view of a network, one that involves moving messages between hosts. It is similar to an information flow model.

NETWORK MESSAGE MODEL

This model is based on a GYPSY model of a network and is provided as a contrast to the Bell & LaPadula model of the previous section. As before, we have n hosts which we will represent by the integers 1, 2,..., n, and a classification level c(j) of each host. We also have a set, M, of messages, consisting of all finite sequences in some alphabet A. The empty message will be denoted by 0. Associated with each message are three functions:

$SOURCE: M \rightarrow \{0, 1, 2, \ldots, n\}$
$DEST: M \rightarrow \{0, 1, 2, \ldots, n\}$
$LEVEL: M \rightarrow L$,

denoting respectively the host that is the source of the message, the host that is to receive the message, and the classification level of the message. (Note that SOURCE(m) = 0 or DEST(m) = 0 if and only if $m = \emptyset$.)

It will be assumed that $LEVEL(\emptyset) \leq LEVEL(m)$, for all $m \in M$. The states of the network will be sets of messages that are waiting to be sent into or out of the network. Specifically,

$$V = \left(\underline{X}_{i=1}^{n} \underline{X}_{j=1}^{k} M \right)^2$$

That is to say, a state v is an 2n-tuple

$$((v_1, \ldots, v_n), (w_1, \ldots, w_n))$$

with each v_i and w_i a k-tuple,

$$v_i = (m_{i,1}, \ldots, m_{1,k})$$
$$w_i = (r_{i,1}, \ldots, r_{i,k})$$

The initial state $v^* = ((v_1^*, \ldots, v_n^*), (w_1^*, \ldots w_n^*))$ is the state with v_i and w_i consisting only of empty messages. The v_i's correspond to the set of messages that host i will send into the net and the w_i's correspond to the sets of messages received from the net.

To construct an automaton, we must define an input set I and a mapping $\rho: V \underline{x} I \rightarrow V$. The inputs will be of three types:

$$I_1 = \{(i, m) \mid 1 \leq i \leq n \text{ and } SOURCE(m) = i\}$$

$$I_2 = \{1, 2, \ldots, n\}$$

$$I_3 = \{1, 2, \ldots, n\}$$

We let $I = I_1 \amalg I_2 \amalg I_3$. The interpretation of these inputs will be made clear through the definition of ρ.

Consider $v = ((v_1, \ldots, v_n), (w_1, \ldots, w_n))$

and consider an input (i,m) of the first type. The first coordinate, i, means that the message m is to be added to host i. Suppose $v_i = (m_{i,1},\ldots m_{i,t}, \emptyset,\ldots, \emptyset)$ where $m_{i,t} \neq \emptyset$. If t = k, the message buffer is full and ρ will not change the state; thus $\rho(v) = v$.

If $t < k$, let
$\tilde{v}_i = (m, m_{i,1},\ldots,m_{i,t}, \emptyset,\ldots, \emptyset)$

Then
$\rho(v) = ((v_1,\ldots,v_{i-1}, \tilde{v}_i, v_{i+1},\ldots,v_n), (w_i,\ldots,w_n))$.

In words, the list of messages in the buffer waiting to be sent are right shifted by one place (if there is room) and the new message is added to the list.

Next consider an input $i \in I_2$. This input will cause the first message placed in the message list for host i to be sent to its specified destination. Specifically, suppose as in the previous case, $v_i = (m_{i,1},\ldots, m_{i,t}, \emptyset,\ldots, \emptyset)$. The destination for $m_{i,t}$ is $DEST(m_{i,t}) = j$. Suppose, $w_j = (r_{j,1},\ldots r_{j,s}, \emptyset,\ldots, \emptyset)$ where $r_{1,s} \neq \emptyset$. If $s \geq k$, the buffer is full and we must define $\rho(v) = v$. Otherwise, the following condition is checked:

$$c(j) \geq \text{LEVEL}(m_{i,t}).$$

That is, the host level must dominate the message level. If this condition fails, $\rho(v) = v$. Otherwise, let
$\tilde{v}_i = (m_{i,1},\ldots,m_{i,t-1}, \emptyset,\ldots, \emptyset)$.
$\tilde{w}_j = (m_{i,t}, r_{j,1},\ldots r_{j,s}, \emptyset,\ldots,\emptyset)$.

Then

$$\rho(v) = ((v_1,\ldots,v_{i-1}, \tilde{v}_i, v_{i+1},\ldots,v_n), (w_1,\ldots w_{i-1},\tilde{w}_i, w_{i+1},\ldots,w_n))$$

The last case corresponds to the command to remove a message from the buffer. Let $j \in I_3$ and assume
$w_j = (r_{i,1},\ldots,r_{j,s}, \emptyset,\ldots, \emptyset)$.

Let $\tilde{w}_j =$
$(r_{i,1},\ldots r_{s,s-1}, 0, 0,\ldots, 0)$, and
$\rho(v) =$
$((v_1,\ldots,v_n),(w_1,\ldots\tilde{w}_j,\ldots,w_n))$.

We have defined, $\rho: V \times I \rightarrow V$ thus defining an automaton. The security theorem that we may prove is that no message will be misrouted and that no message will go to a host having an operating classification lower than the message level. These security properties are formalized through the following properties.

Nondisclosure Property

A state $v = ((v_1,\ldots,v_n), (w_i,\ldots, w_n))$ with $w_j = (r_{j,1},\ldots,r_{j,k})$ satisfies the nondisclosure property if and only if for each w_j, $j = 1,\ldots,n$ and each $s = 1,\ldots,k$, $c(j) \geq \text{LEVEL}(r_{j,s})$.

Nonmisrouting Property

A state $v = ((v_1,\ldots, v_n) (w_1,\ldots,w_n))$ with $w_j = r_{j,1},\ldots r_{j,k})$ satisfies the Nonmisrouting Property if and only if for each i, $k = 1,\ldots,n$ and each t,s, $t = 1,\ldots,k$, $s = 1,\ldots k$,

$$\text{SOURCE}(m_{i,t}) = \begin{cases} i \text{ if } m_{i,t} \neq \emptyset \\ 0 \text{ if } m_{i,t} = \emptyset \end{cases}$$

$$\text{DEST}(r_{j,s}) = \begin{cases} j \text{ if } t_{j,s} \neq \emptyset \\ 0 \text{ if } r_{j,s} = \emptyset. \end{cases}$$

Theorem. Let $\rho: V \times I \rightarrow V$ be defined as above. Let v* be an initial state of empty messages. Then for any sequence $i = i_1, i_2,\ldots,$ of inputs, $\rho^*(v^*,i)$ satisfies Nondisclosure and Nonmisrouting.

Proof. The proof is by induction on the length of the input sequence. We first show that it holds for sequences of length

0. This is trivial as the initial state clearly satisfies Nonmisrouting and Nondisclosure.

Suppose $i = i_1,\ldots,i_q$. Let $v = \rho^*(v^*,i_1,\ldots,i_{q-1})$. Then v satisfies Nondisclosure and Nonmisrouting by the induction hypothesis. There are three cases:

$i_q \in I_1$, or $i_q \in I_2$, or $i_q \in I_3$.

Case 1

$i_q \in I_1$

In this case, SOURCE(m) = i by definition of I_1 and

$$\rho(\ \rho^*(v^*, i_1,\ldots,i_q)\ ,i_q) = ((v_1,\ldots,\tilde{v}_i,\ldots,v_n), (w_i,\ldots,w_n)).$$

Nondisclosure and Nonmisrouting hold in v_j, $i \neq j$, and all w_i by induction. Now $\tilde{v}_i = (m, m_{i,1},\ldots, m_{i,t}, \phi,\ldots, \phi)$ or is unchanged. Since SOURCE(m) = i, Nonmisrouting holds.

Case 2

$i_q \in I_2$

Going back to the definition of , the only things that change are in $v_i = (m_{i,1},\ldots,m_{i,t}, \phi,\ldots, \phi)$ and $w_i = (r_{j,1},\ldots, r_{j,s}, \phi,\ldots, \phi)$. The Nondisclosure and Nonmisrouting conditions hold in v_i since a message is replaced by an empty message. On the other hand, w_j is changed to $\tilde{w}_j$ only if $c(j) \geq$ LEVEL($m_{i,t}$) which maintains Nondisclosure. As DEST($m_{i,t}$) = j and SOURCE($m_{i,t}$) = i by definition, Nonmisrouting still holds.

Case 3

This is trivial and is left as an exercise for the reader. □

In the previous sections we have defined two network models; one an access control model based on the Bell and LaPadula model schema and the other based on message flow between hosts of the network. Both models assume that connections between hosts are essentially pipelines and that once established, information flows directly between the hosts without error. Except for "error-freeness," these models should apply to any real network. Since all the switches, gateways, and network controllers may be interpreted as hosts of one sort or another, albeit with limited capabilities. Neither model by itself, however, captures all of the features one wants to model. It is necessary to unite the two models in some fashion. In the next few paragraphs, we will discuss a method by which this unification can take place.

In order to capture both the access control and the message buffers in the system state, we let the new network state be the cartesian product $V_1 \times V_2$ where V_1 and V_2 are the system states for the Bell and LaPadula and the message models defined previously. Thus, a state will include the access controls and the pending messages. The input set I of the automata model of the network could be converted into requests and simply included in R_{Net} in the obvious way. However, this will not really couple the two models together. What needs to be done is to redefine the interhost access requests to utilize the messages. That is, a connect or log on request will require the sending of a special "connect" or "logon" message between the hosts. In the same way, each of the accesses will have an associated message that must first be distributed to the other hosts. A subsequent paper will explore these issues in more detail as well as begin consideration of the error issues.

REFERENCES

[1] Bell, D. Elliott and LaPadula, Leonard J., "Secure Computer Systems: Unified Exposition and Multics Interpretation;" MTR-2997; The MITRE Corporation, Bedford, Massachusetts; 1 March 1973.

A TWO-LEVEL SECURITY MODEL FOR A SECURE NETWORK

Janice I. Glasgow
Glenn H. MacEwen

Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada, K7L 3N6

ABSTRACT

A formal security model for the SNet multi-level secure distributed system is described. The model comprises two parts, an abstract model based on history sequences representing message flow in the system, and a concrete model that extends the abstract model with operations that change user states and with explicit labelling of messages. The proof of consistency between the abstract and the concrete models is outlined. Current work to specify the system using the Lucid language, to prove the consistency of these specifications with the model, and to transform these specifications into Concurrent Euclid is not described.

1. INTRODUCTION

This paper describes part of current work toward the design and implementation of a multi-level secure distributed system called SNet, an early version of which was first described in [MacEwen84]. Some initial work to define a security model for SNet, and to formally specify the system using the Lucid/dataflow approach, was presented in Gaithersburg in September 1984 [Glasgow84]. The model presented at that time was a very concrete model, and one which was more complex than we would have liked. Subsequent consideration, as well as some discussion at the Gaithersburg conference, led us to attempt to refine the model into two levels of abstractness. This paper discusses security models in general, the various problems of information flows in SNet, and describes the abstract and concrete security models for SNet.

We are currently using Lucid to specify the system and are verifying the correctness of the specification with respect to the model. The implementation of SNet is in Concurrent Euclid. We are currently investigating the systematic transformation of Lucid specifications into Concurrent Euclid code. This specification and implementation work is the subject of papers under development.

2. SECURITY MODELS

A security model is a special component of a system's requirements that describes a safety property. That is, it specifies that certain things related to security must not happen during the operation of the system. In other words, it is a set of constraints on the function of the system. Another characteristic that distinguishes the security model from other functional requirements is the fact that it must be described in formal mathematical terms due to the requirement for formal verification of security properties.

2.1 Components of a Security Model

A security model may comprise several levels, each describing the constraints in successively more detailed terms. The upper level abstract model describes the security constraints in a very simple way that can be accepted as the basic definition of security for the system. Successively less abstract levels add detail and representation concerns that are necessary to relate the model to the functional specifications. The constraints on each level must be shown to imply the constraints on the next more abstract level. The lowest least abstract level is called the concrete model.

Figure 2.1 shows the relationship of these model levels with the functional specifications of a system. Functional specification starts with a requirements specification which is usually an informal English description of the behavior of the system. Some formal modeling may also exist at this level of specification but this is only likely to be used to describe precisely some aspects of the behavior. The first level of specification that is entirely expressed in formal terms is usually called the design specification. This may also comprise sub-levels starting with the top-level specification as shown in the diagram. In a similar way as with the security model, successively less abstract levels add detail and representation concerns.

The important difference to realize is that the model levels attempt only to constrain the function of the system while the specification levels attempt to fully describe the function of the system. Therefore they have quite different objectives and may be expressed in quite different languages. However, at some point they must be related. This is done at the lowest model level and the highest design specification level. The objects in the model must be associated with objects in the design and the model constraints applied to the design. The design specifications must then be shown to conform to the constraints.

It is therefore important that, although the abstract model may be expressed in a language not closely related to the design the concrete model must be easily related to the design. One may include some mechanisms in the concrete model that are not directly concerned with the constraints. This happens in the effort to produce a concrete model that can easily relate to the design specification.

It is reasonable that the same model can be used for more than one set of specifications - for the same system, or for different systems. In this case, also illustrated in Figure 2.1, different concrete models can be derived from a common upper level model. In this way, the same basic notions of security can be applied to both systems while a different concrete model is used for each.

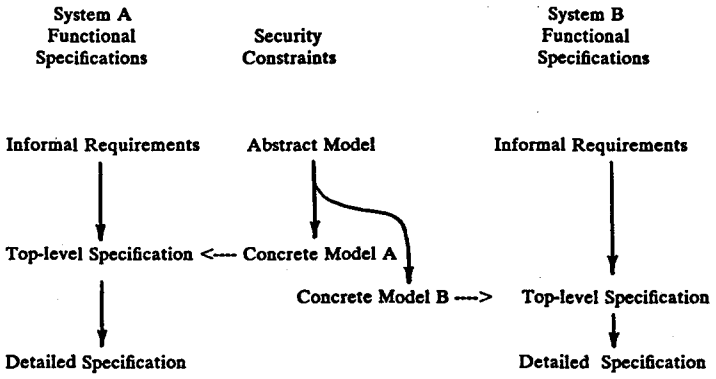


Figure 2.1 Relationship of Security Model to Specifications

The development of the security model as a series of less abstract models can assist in developing the specifications. Since one wants to be able to verify the correctness of the top-level specification with respect to the concrete model, it is essential that that two are stated within a common framework. Consequently, with each security model level one can develop an associated partial specification to reflect the structure and objects in the model. In this way, when the top-level specification, which should be complete, is produced it is easier to associate it with the concrete model, and the verification should be more tractable.

Furthermore, as one develops each partial specification one can formally verify its consistency with the associated model level. In this way, the correctness of the top-level specification can be established in steps concurrently with the development of the concrete model. This is, in fact, what is done in this paper; a Lucid partial specification is shown to be consistent with the abstract model for SNet.

2.2 Security in SNet

SNet is a multi-level secure system in which users at terminals and host computers exchange messages (Figure 2.2). The essential idea is that an untrusted host can be used to store information of a common security level. Trusted multi-level hosts can also be accessed from the same user terminals. Such trusted hosts may perform such special security functions as secure downgrading.

A user can establish a set of active virtual circuits from her/his terminal to any number of hosts. At any time, however, only one of these circuits can be connected and when so connected there is transparent communication between the terminal and the associated host. The user specifies her/his security level for each circuit; for the network to allow bi-directional communication over a circuit to an untrusted host this level must be the same as that of the host since otherwise flows are possible only in one direction.

A user can, for example, connect to host A to logon, switch to host B to logon, and then switch back to A to initiate a file transfer from A to B. Hosts can communicate over the network at any time, and in any way permitted by the security enforcement in the network. Host-to-host protocols must be based on uni-directional communication only however, so that hosts of different levels can transfer data from a lower level to a higher level. This means that the user doing a transfer as just described must switch back to host B to check that the file was received correctly. All bi-directional protocols to perform flow control and error detection and recovery must exist within trusted components of the network.

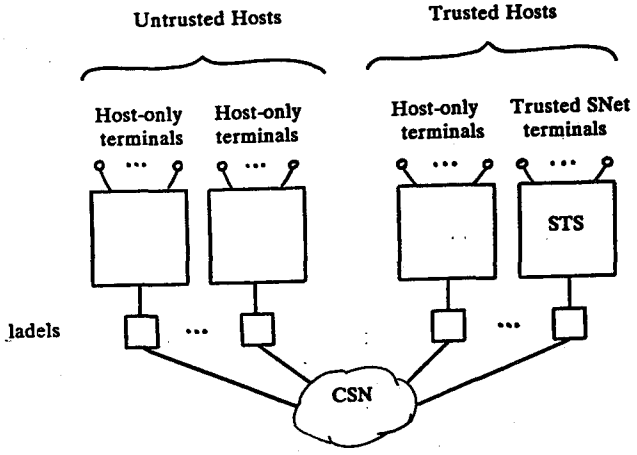


Figure 2.2 SNet Architecture

As suggested by Figure 2.2, the virtual circuit mechanism exists in a trusted Secure Terminal Server (STS) which looks to the rest of the network just like another

trusted host. All hosts are connected to an untrusted Communication SubNetwork (CSN) by a secure network interface unit called a Labeller/Delabeller (ladel). All communication is in message units. Each message, as it is sent, is labelled with the security level of the information in the message. This label is used at the destination to determine if the message can be passed on to the recipient according to the common lattice model of secure information flow. Users and trusted hosts can determine the level in messages that they send while untrusted hosts have a fixed level attached to all outgoing messages. The communication subnetwork that supports the message traffic is not trusted. This may result from having untrusted switching nodes or having untrusted hosts on the network capable of monitoring messages and transmitting messages without going through the secure labelling mechanism.

There are two kinds of information flows that a security model for SNet must address: storage flows and covert flows. Enforcement of the first involves three security requirements. Enforcement of the second involves four requirements. These seven requirements are discussed in turn below.

Storage flows, of course, occur in the message traffic between hosts and users. The labels on messages can serve as the basis for an enforcement mechanism to limit such flows. First, a transmitted message must be properly labelled depending on whether or not the sender is trusted. Second, the message flow must be checked at the recipient to conform with the security policy embodied in the lattice of levels. For this to work, of course, the messages, data and associated level, must pass through the network unaltered. This, then, constitutes the third requirement: the integrity of each transmitted message.

Covert flows can occur because messages, along with the data and the level must of necessity carry other control information such as a network address. Consider, a message sent from host A at level L_A to host B at level L_B while host C at level L_C is sending a stream of messages to host D at level L_D where $L_C \leq L_D \leq L_A \leq L_B$. An untrusted agent in the network could observe control information in the message going from A to B and encode this information in the message stream going from C to D. This results in a flow from A to D contrary to the lattice model.

How can such information be encoded? The agent could re-order messages is a way detectable by the recipient or it could save a copy of the message stream and resend identical messages in a predetermined pattern. The fourth and fifth requirements, then, are for uniqueness of messages, i.e. a non-duplication requirement, and for order preservation of messages.

A third way in which such coded information could be sent is for the agent to redirect messages from some other stream and send them to D in some predetermined pattern. Of course, such message stealing may be detectable but this is dependent on the application. In SNet such message loss would likely appear as a file transmission error and require resending of the file involved. If this did not happen often then it would not appear unusual. The sixth requirement, then, is for routing correctness.

Finally, if it were possible to create legitimate messages then they could be used directly to transmit covert information. This leads to a requirement for authentication of messages. This means that any message received is one that was sent by a legitimate user or host, and is equivalent to disallowing the creation of messages in the network.

Of course, message deletion can also be used to encode information for the purpose of transmitting it out of the untrusted subnetwork. However, a requirement that there be no lost messages seems impractical to consider for at least two reasons. First, it is a severe requirement to impose on any network. Second, it may conflict with some of the other requirements. For example, reasonable implementations of flow control and order preservation may involve message deletion. Since our model is one that only states constraints that can be enforced, we conclude that this covert channel cannot be handled at this level. It should, though, be addressed at the implementation level with some monitoring mechanism that detects unexpected message loss.

To summarize then, the SNet security model must address these seven requirements: labelling, message flows, integrity, uniqueness, order preservation, routing, and authentication.

3. ABSTRACT MODEL

To simplify the process of developing a concrete model for our secure system we introduce a level of abstraction. This intermediate model defines high-level constraints for the system.

3.1 Description of Abstract Model

Here we attempt to express the fundamental notions of security in SNet in as simple and precise a way as possible. The abstract model contains the following:

```
A set S of subjects
A set D of data
A set M of messages
A set E of events
A partially ordered set H of histories
A partially ordered set L of levels
A mapping Trusted: S -> Boolean
A mapping Subjectlevel: S x H -> L
A mapping Maxlevel: S -> L
A mapping Messagelevel: M -> L
A mapping Destination: M -> S
```

A mapping Data: M -> D
A mapping Sender: M -> S
A mapping Newhistory: H x E -> H

We explain these objects in detail in the following definitions and in the rest of the paper. However, an informal explanation of their interpretation is helpful in understanding the model. The set S of subjects comprises both users and hosts. Both communicate via messages containing data. An event is the transmission of a new message or the receipt of a previously transmitted message. A history is a set of pairs where each element of a pair is an infinite sequence containing respectively all transmitted and all received messages by one subject. So a history describes at any particular time all events that have occurred.

The function Trusted is constant and essentially specifies, for any subject, whether or not it can determine the label on messages that it sends. All users and some hosts are trusted. For untrusted hosts, the label on transmitted messages is determined by the function Maxlevel. In the implementation, Maxlevel and Trusted are secure external inputs to the ladel devices; If Trusted is false then all transmitted messages are labelled with Maxlevel. If Trusted is true, the message label is provided by the host but may not be greater than Maxlevel. For an STS, of course, Maxlevel will be the highest level.

The function Subjectlevel describes the message label that is provided by trusted subjects. This is needed because users can change their current level as they switch from one virtual circuit to another since each virtual circuit has a distinct associated user level. For trusted hosts, there is no restriction on Subjectlevel however, since a trusted host is assumed to correctly provide a label for every transmitted message.

The functions Messagelevel, Destination, Data, and Sender simply define the four fields of a message.

Finally, Newhistory defines an event as the act of appending a message to one of the infinite sequences in a history.

The following defines these objects more formally.

Definition 3.1 Message Equality

A message in the model consists of four values: the name of the subject who sent the message (sender), the level of the sender when the message is transmitted (messagelevel), the subject to whom the message is being sent (destination) and the data being sent. Since messages are not unique we define the notion of equality of messages in terms of these four characteristics:

For all messages x and y in M, x = y iff
Messagelevel(x) = Messagelevel(y)
Destination(x) = Destination(y)
Data(x) = Data(y)
Sender(x) = Sender(y)

Definition 3.2 Message Sequence

For every subject s_i in S, we consider t_i and r_i as infinite sequences of values representing transmitted and received messages respectively. We denote a message sequence as an ordered sequence $x_i = \langle x_{i1}, x_{i2}, \ldots \rangle$ where message x_{ij} is an element of set M for all $j \geq 1$.

Message sequences can be considered as history sequences where x_{ij} is the jth value for sequence x_i.

Definition 3.3 Network History

We define a history for a network to be a set of ordered pairs containing the transmitted and received message sequences for all subjects s in S. H is the set of all possible histories for the network.

For any history h, we say that x_i is a message sequence in history h if x_i is an element of one of the ordered pairs in the history. Although x_i is an infinite sequence, for any h it contains only a finite number of defined messages, i.e. $x_i = \langle x_{i1}, x_{i2}, \ldots, x_{in}, \perp, \perp, \ldots \rangle$. In this case x_i contains n defined messages, the remaining messages are $\perp$ (undefined). We define the mappings Messagelevel($\perp$) = Sender($\perp$) = Destination($\perp$) = Data($\perp$) = $\perp$. Thus $\perp$ is contained in the sets S, M and L. To preserve the partial ordering of L we say that $\perp \leq l$, for all l in L.

Definition 3.4 Event

There exist two types of events that occur in the abstract model for the network. For any defined message m in M and any subject s_i in S, a send event send(s_i,m) denotes the transmission of message m by subject s_i. This results in a new message being added to the sequence t_i. Similarly a receive event" receive(s_i,m) denotes that subject s_i received message m from the network and results in sequence r_i being augmented. This change in sequences is described more fully by the following definition.

Definition 3.5 Newhistory

We define the mapping Newhistory that maps a history and an event onto a new history as follows:

Newhistory(send(s_i,m),h) = h'
where h' is the same as h except if sequence t_i in h = $\langle t_{i1}, \ldots, t_{in}, \perp, \perp, \ldots \rangle$ then t_i in h' = $\langle t_{i1}, \ldots, t_{in}, m, \perp, \perp, \ldots \rangle$.

Newhistory(receive(s_i,m),h) = h'
where h' is the same as h except if sequence r_i in h = $\langle r_{i1}, \ldots, r_{im}, \perp, \perp, \ldots \rangle$ then r_i in h' = $\langle r_{i1}, \ldots, r_{im}, m, \perp, \ldots \rangle$.

Thus, applying Newhistory to a given history and event results in appending one of the transmitted or received history

sequences with the message specified in the given event.

Definition 3.6 Multisets

A multiset [Knuth81] is like a set, but it may contain identical elements repeated a finite number of times.

For any two multisets M and M', we say that M ⊑ M' (multiset subset) iff for any element x that occurs a times in M, x occurs b times in M' such that a ≤ b.

The concept of a multiset is used to collect defined messages from a set of message sequences. Since these messages are not unique it is necessary to record all occurrences of identical messages.

Definition 3.7 Collections

We define the function Collect as a mapping from a set of message sequences to a multiset of messages as follows:

$Collect(x_1,x_2,\ldots,x_n) = M$

where M is the multiset that contains a occurrences of element x_{ij} (for all i, $1 \leq i \leq n$) if x_{ij} is a non-nil element occurring a times in sequence x_i.

The notion of a collection of messages is used to prove the constraints of authenticity, integrity and uniqueness. By comparing the collection of messages that are transmitted by all subjects to those messages that are received we can specify that no messages were changed or created in the untrusted network.

Definition 3.8 Partial Orderings

a) Levels
The partial ordering on levels is denoted ≤. Alternatively we use Dominates(l2,l1) to denote $l1 \leq l2$.

b) Sequences
For any two history sequences $x = \langle x_1,x_2,\ldots\rangle$ and $y = \langle y_1,y_2,\ldots\rangle$ we say that $x \leq' y$ (where $\leq'$ denotes partial ordering on sequences) iff for all $i \geq 1$, $x_i = \perp$ or $x_i = y_i$.

c) Histories
We define the partial ordering ≤" on histories of the network as follows:

If h1 and h2 are two histories in H such that
$h_1 = \{(r1_1,t1_1),\ldots,(r1_n,t1_n)\}$ and
$h_2 = \{(r2_1,t2_1),\ldots,(r2_n,t2_n)\}$ then

$h1 \leq'' h2$ iff
$r1_i \leq' r2_i$ and
$t1_i \leq' t2_i$ (for all $i \geq 1$).

3.2 Security Constraints for Abstract Model

The following constraints must hold for all histories $h = \{(r_1,t_1),\ldots,(r_n,t_n)\}$ in H of a secure network:

C0: Labelling

If send(s_i,m) is an event that occurs in history h (i.e. it produces a new history h' from h) then it must be the case that:

not Trusted(s_i) implies
Messagelevel(m) = Subjectlevel(s_i,h) = Maxlevel(s_i)
Trusted(s_i) implies
Messagelevel(m) = Subjectlevel(s_i,h) ≤ Maxlevel(s_i)

C1: Message Flow

If receive(s_i,m) is an event that occurs in history h then it must be the case that:

Dominates(Maxlevel(s_i),Messagelevel(m))

C2: Authenticity, Integrity and Uniqueness

$Collect(r_1,\ldots,r_n) \subseteq Collect(t_1,\ldots,t_n)$

C3: Routing

If m is a message in some sequence r_i ($1 \leq i \leq n$) then
Destination(m) = s_i

C4: Order Preservation

If $\langle r_{j1},r_{j2},\ldots,r_{jn}\rangle$ is any ordered subsequence of some sequence r_j (i.e. the subsequence preserves the order of the original sequence) such that $Sender(r_{j1}) = Sender(r_{j2}) = \ldots = Sender(r_{jn}) = s_p$ then there exists an ordered subsequence $\langle t_{k1},t_{k2},\ldots,t_{kn}\rangle$ of sequence t_p such that $t_{ki} = r_{ji}$ for all i, $1 \leq i \leq n$.

This constraint is also dependent on the constraint of sender validity. That is, for all transmitted sequences t_i, if t_{ij} is a defined message in t_i then $Sender(t_{ij}) = s_i$.

4. CONCRETE MODEL

On this level of the model we want to relate the concepts in the abstract model to the operations performed at the externally visible interface of the system by the external agents: users and host computers. In doing so, a general question arises: What is in the model and what is in the specification? In other words, there is a tendency to refine the model in ways that do not alter or add to the basic security constraints and that do not affect the definition of the operations. In both these cases, the refinements belong properly in the specification and not in the model.

The following refinements to the abstract model add some security rules related to user/host operations, define these operations, and relate the operations to the sequences defined in the abstract model.

4.1 Description of Concrete Model

The abstract model is extended to contain the following:

A set U of users
A set C of computers
A set F of user messages
A set G of computer messages

A set UC of valid user commands
A set V of all possible sets of virtual circuits
A set US of possible user states
A mapping Newstate: US x E -> US

The following definitions more fully describe these extensions to the model:

Definition 4.1 Subject Partitioning

The set S of subjects is partitioned into two disjoint sets: computers (C) and users (U), i.e.:

$$S = U \cup C, \quad U \cap C = \Phi$$

where $\forall u \in U$, Trusted (u) = true.

Definition 4.2 User Messages

User input is modelled as an infinite history sequence of user messages where the set of all such messages is denoted F. The meaning of a user message is determined by the mappings:

Command : F -> UC
CommandDest : F -> C
CommandLevel : F -> L
CommandData : F -> D

where UC = {usend, open, close, connect, disconnect}, the set of valid user commands.

Definition 4.3 Computer Messages

Computer input is modelled as a sequence of computer messages where the set of all possible computer messages is denoted G. The meaning of a computer message is determined by the mappings.

ComputerDest : G -> S
ComputerLevel : G -> L
ComputerData : G -> D

Definition 4.4 History

We extend the notion of history for a network to include user message sequences and computer message sequence. For each subject s_i we denote such a sequence as o_i, where o_i is a user message if s_i is a user and o_i is a computer message if s_i is a computer. A history, thus, comprises a set of triples:

$\{(o_1,t_1,r_1),\ldots,(o_n,t_n,r_n)\}$.

Definition 4.5 User State

Each user has an associated local state, one component of which is the set of virtual circuits that have been established between that user and one or more hosts. A virtual circuit is a pair denoted (c,l), $c \in C$, $l \in L$ and a set of circuits is denoted $v = \{ (c,l) \mid c \in C, l \in L\}$. The set of all possible sets of virtual circuits is denoted V.

The complete user state is a pair consisting of a set of circuits v and a circuit (c,l). The pair (c,l), which may be $(\perp,\perp)$, represents the connected virtual circuit between the user and one computer c, with l being the level to be associated with the user for that circuit. The set of valid user states are constrained by the function Newstate defined later in this section. The value of a user u's state for a given history h is expressed as a mapping Userstate(u,h).

Definition 4.6 Subjectlevel

The mapping Subjectlevel from the abstract model is defined to include a dependence on the users' message sequences. That is, a user can alter his/her current level (in a way described below) and the value of the current level is determined, for any history h, from the user history sequence. Since the current level is a component of the local state (which in turn is a function of the history) Subjectlevel is defined on this basis.

For any $u \in U$ and $h \in H$,
if Userstate(u,h) = (v,(c,l)) then
Subjectlevel(u,h) = l

For the case of computers we allow Subjectlevel, for any history, to be unspecified.

For any $c \in C$ and $h \in H$
Subjectlevel(c,h) is determined by c.

Definition 4.7 Command Event

The concrete model adds a third type of event, the command event, denoted comm(s_i,o) for $s_i \in S$, $o \in F$. The mapping Newhistory is refined as follows:

Newhistory(comm(s_i,o),h) = h'

where h' is the same as h except if sequence o_i in h = $\langle o_{i1},\ldots,o_{in},\perp,\perp,\ldots\rangle$ then o_i in h' = $\langle o_{i1},\ldots,o_{in},o,\perp,\perp,\ldots\rangle$

Thus, applying Newhistory to a given history and command event results in appending one of the user or computer messages to the associated user or computer message sequences.

Definition 4.8 Newstate

Certain user messages can cause a change in the associated user state. We represent this as a function:
Newstate: US x E -> US.
The initial value of a user state is the pair $(\{\}, (\perp,\perp))$. Given any user state st = (v,(c,l)) and a command event comm(u,o) we generate the next state st' = Newstate(st,comm(u,o)) = (v',(c',l')) where v',c',l' are defined as follows:

If Command(o) = open then
  $v' = v \cup$ (CommandDest(o),CommandLevel(o))
  $c' = c$
  $l' = l$

If Command(o) = close then
  $v' = v -$ (CommandDest(o),CommandLevel(o))
  $c' = c$
  $l' = l$

If Command(o) = connect and $(c,l) = (\perp,\perp)$ and (CommandDest(o),CommandLevel(o)) $\in v$ then
  $v' = v$
  $c' =$ CommandDest(o)
  $l' =$ CommandLevel(o)

If Command(o) = disconnect then
  $v' = v$
  $c' = \perp$
  $l' = \perp$

For all other user messages o, the state remains unchanged, i.e. $v' = v$, $c' = c$ and $l' = l$.

4.2 Constraints for the Concrete Model

We add the following constraints to those already given in the abstract model:

C5: User Subjectlevel

For all users u such that Userstate(u,h) = (v,(c,l))
  $l \leq$ Maxlevel(u)

C6: Valid Messages

For any history h in H and user u in U send(u,m) is a send event in history h iff there exists a previous history h' in H ($h' \leq'' h$) such that:

Userstate(u,h') = (v,(c,l)) for some v in V and (c,l) in v
comm(u,o) is a unique command event occurring in h' where:
  Command(o) = usend
  Data(m) = CommandData(o)
  Destination(m) = c
  Messagelevel(m) = l

For any history h in H and computer c in C send(c,m) is a send event in history h iff there exists a previous history h' in H ($h' \leq'' h$) such that:

comm(c,o) is a unique command event occurring in h' where:
  CommandData(o) = Data(m)
  CommandDest(o) = Destination(m)
  If Trusted(c) and Dominates(Maxlevel(c), CommandLevel(o))
    then Messagelevel(m) = CommandLevel(o)
    otherwise Messagelevel(m) = Maxlevel(c)

Constraint C5 specifies that the level for any connected virtual circuit must be dominated by the user's maximum level.

Constraint C6 relates command events to send events. Every send event must be the result of a valid command event. In the case of users a send event corresponds to a command event with user command usend. For both user and computer commands the message resulting from the send event must have the same data and destination as the original command event. A send event for a user must also have a destination that corresponds to the connected virtual circuit at the time it was sent. Constraint C6 also extends and replaces constraint C0 of the abstract model by specifying that the level of any message transmitted by a user must be the current Subjectlevel for that user as given in the user state. In addition, trusted computers can determine their Subjectlevel and thereby specify the label on messages, while untrusted computers have their messages labelled with their Maxlevel value.

5. CONSISTENCY OF MODELS

In previous sections two models for the secure system were presented: the abstract model and the concrete model. The concrete model was developed by extending the abstract model to consider operations externally visible to the user.

To show that the two models are consistent we need only verify that the constraints of the more refined model imply the constraints of the abstract version. Since the concrete model constraints are in fact a superset of the abstract model's it is clear that this holds, i.e.

C0 & C1 & C2 & C3 & C4 & C5 & C6
  -> C0 & C1 & C2 & C3 & C4

It is in fact possible to remove constraint C0 from the concrete model since we can prove that this constraint is implied by constraints C5 and C6, that is C5 & C6 -> C0.

The construction of a concrete model for the secure system consisted of a two-stage process. This allowed us to consider and verify more abstract constraints before looking at the details of the model. Although this also implied two stages of proofs as well as the need to demonstrate the consistency of the models, it simplified the individual proofs and, as discussed above, the consistency proof is straightforward.

6. CONCLUSIONS

It has proved to be surprisingly difficult to express the notions of security for SNet in the security model. And the model still seems to be more complex than we would prefer. However, the result appears, at this point in the development, to facilitate the formal specification in Lucid. We do not anticipate that the proofs required will be difficult. Furthermore, we hope to be able to generate procedural code from the Lucid specifications

which can be used to verify the implementation. This work will be reported in a future paper.

REFERENCES

[Glasgow84]
J. Glasgow, F. Ouabdesselam, G.H MacEwen, T. Mercouris, "Specifying Multi-level Security in a Distributed System", NBS/DOD Computer Security Conference, Gaithersburg, MD, September, 1984.

[Knuth81]
D.E. Knuth, "Seminumerical Algorithms" The Art of Computer Programming, Vol. 2, Second Edition, Addison Wesley, 1981.

[MacEwen84]
G.H. MacEwen, Z. Lu and B. Burwell, "Multi-Level Security Based on Physical Distribution", IEEE Symposium on Security and Privacy, Oakland, April, 1984. Also presented at NBS/DOD Computer Security Conference, Gaithersburg, MD, September 1984.

NETWORK SECURITY ASSURANCE*

Marvin Schaefer
DoD Computer Security Evaluation Center
Fort George G. Meade, Maryland

D. Elliott Bell
Trusted Information Systems, Incorporated
Glenwood, Maryland

Abstract. The issue of assuring network security is addressed. It is argued that the principles underlying assurance of classical computer security (conceptual, formal, and implementational) apply directly in the network context, providing either a full solution or directions for further research.

INTRODUCTION

It is the premise of this note that there is little intrinsic difference in the assurances required of a monolithic trusted operating system, a distributed trusted operating system, and a trusted network. In each case, the system is assumed to enforce a precise statement of a system security policy. A system security policy is interpreted to mean a statement that characterizes all permissible modes of access between each subject and each object on the system: it is an access control policy. And in each case, modeling, careful system design architecture, and formal verification are used to provide assurance about fidelity to that policy.

The DoD Trusted Computer Systems Evaluation Criteria [8] (a. k. a. the TCSEC) has very little that ties it tightly to computers of the classical kind. True, the casual reader can get the impression that much is computer-specific, but there are few places where we are convinced that it is so. The identification of 'user' with a person is the most glaring. The specificity of what must be contained in the various audit records is another example. But based on our personal experiences with Blacker and the I-S/A AMPE, we do not think that the TCSEC deals badly with the network concept.

Although the assurance issues in network security cover a wide gamut of topics, we have concentrated on three of them. The two most obvious areas of concern are modeling and formal verification. Related are the several facets of system design and architecture that justify and support the application of higher level abstractions in constructing trusted systems. We will address these three areas -- modeling, design, and formal verification -- adding tangential asides as appropriate.

*Nothing in this paper is to be taken as having been endorsed by any government agency. References to BLACKER and I-S/A AMPE are to be understood as the authors' personal conclusions deriving from technical discussions of the projects. The points of view remain personal rather than official.

MODELING

What properties should network security modeling address? The traditional computer security foci are compromise, need-to-know, and the derivative problem of information flow, all within the context of the Reference Monitor Concept (see [1]). Two widely mentioned network problems are information integrity and denial of service. We will address all of these properties, but (the second fundamental question) at what level of abstraction?

Level of Abstraction

From a modeling standpoint, networks need to be addressed both in the large and in the small. In the case of Blacker, that meant that the system and each of the three "network security" components had to be subjected to the Policy-to-Model-to-FTLS and DTLS-to-Code correspondence exercises. The hard problem was convincing ourselves that the system view and the components' views fit together in the "right" way. We are convinced that it can be done and with no less assurance than is the case for monolithic systems architectures. The formal details need to be worked out both for specification and verification since it needs to be shown that (a) the model has been properly interpreted in the system specification, (b) that the individual trusted components interpret the model appropriately and consistently, and (c) that the components together correctly implement the model.

Subjects and Objects

Underlying discussions of any network security property at either level of abstraction is the first difficult question, that of defining the subjects and objects in question. A related question is that of determining how (and if at all) subjects are to interact.

One popular operating system view has been that every subject is also an object. This leads to the possibility of deriving an access control matrix in which the potential modes of access between every pair of subjects is explicitly exposed. The full rules of the formal security policy model have been applied in such cases to treat security levels of subjects and subjects considered as objects in order to determine whether communication would be authorized or not. Operating system models

have permitted asymmetry in the access control matrix to allow for the strict upward flow of data.[1]

A different view in operating system security models has been that all subjects are formal <process, domain> pairs, and the domain of each subject contains the set of objects that can be named and addressed by the subject. For notational convenience, let dom(A) be the domain of subject A. In this view, subject A can communicate to subject B providing there exists some object in both dom(A) and dom(B) that can be written by A and read by B. The permissible communications between subjects can be derived from the access control matrix between subjects and objects. It is not necessary to build a special matrix in which subjects are paired with subjects.[2]

In the context of classified processing, the primary concern is the security clearance of the end recipient of a transmitted communication, and sometimes the clearance of the originating transmitter of a communication. A secondary consideration becomes the identity of the two participants in the communication.[3] In this sense, matters become much more complex unless the definition of subject is made quite precise. As can be seen, the need for precision on subjects is underscored by the need to be precise about whether subjects are to be interpreted as though sometimes objects or as <process, domain> pairs.

Note that the distinction may not be a strict dichotomy. From the most abstract point of view, subjects can be structureless and rightly c2 Can be considered as having object-ness. When one refines towards an implementation, the abstractions need to be related to the higher level abstractions to benefit from the organizing principles and insight of the higher level. If processes are used, interpretation of <process, domain> pairs as subjects is perfectly reasonable and does not even contradict the subject-as-object view. In particular, "process" as "program in execution" makes clear that "process" represents the active nature of a program; the passive nature (particularly of a quiescent program) cannot reasonably be related to other than an object. What then is become of subject-as-object? The <process, domain> subjects represent the active part of subjects, while the passive part of subjects, the subject-as-object part, is represented by program-artifact objects.

---

[1]It has been observed that this asymmetry is acceptable on a single mainframe because of the relative reliability of the interprocess communication mechanism, while it would not be acceptable on an unreliable medium such as the ether of a network. This concern will be addressed below.

[2]The same concern about access asymmetry has been voiced for this interpretation.

[3]This is of concern because an authorized and properly cleared recipient of classified data cannot view the information except in an appropriatly protected (cleared) environment.

In the network security context, there appears to be room for explicit manifestations of both points of view. Those who place hosts on networks, particularly those who place their hosts on networks, tend to take the position that they need to have explicit control over the selection of those other hosts with which they will authorize communications. These concerns appear to be distinct from security level considerations -- they relate to the identity of each individual host. In some cases, the individual access control concerns are extended to the process on the individual host, or the kind of service being provided, or the individual user on whose behalf the service being performed by the process on the individual host is acting, and so on.

However at the highest level of abstraction, it seems most appropriate not to limit all subjects to be <process, domain> pairs. By so doing, one can consider the hosts as subjects, and the *liaisons*[14] as objects. A pair of hosts is permitted to intercommunicate providing there exists at least one *liaison* that can be both written and read by each of the hosts. Unidirectional communication between hosts can be treated by the case in which there exists at least one *liaison* that can be written by one of the hosts and read by the other.[4] The remaining finer-granularity access controls within network components (themselves traditional computing complexes) can be interpreted under more traditional views.[5]

By its nature, then, a network seems to require modeling attention both at the system level and at the component level for those components that evince a high degree of computing generality. While the interpretations of subjects and objects will be different at the two levels of abstraction, it is not really any change at all to allow the strict <process, domain> pair interpretation at the component while allowing an analogous but

---

[14]In this discussion, we use the French word "*liaison*" to refer to the collection of network proximate connections used to effect communication between two hosts. In this sense, a *liaison* may be instantiated at any time by a specific combination of distinguished computer ports, access lines, encryption devices, switching devices, and so on, forming a path between the two ends.

[4]In the case of the AUTODIN system, reliable unidirectional communication is achieved by interpreting the *liaisons* as the AUTODIN network, which reliably receives the message from the transmitting host and reliably delivers the message to the receiving host. The reliability of receipt and delivery is achieved through a bidirectional communication between the *liaison* and the individual host. See TCB Communication below.

[5]The complexity of network-subjects due to the uncertain nature of "hosts" on a network is discussed in The Variety of Network Subjects below.

not identical interpretation at the system level.

Scope of Security Properties

Return finally to the question of what network properties should be addressed. We use as our framework the Reference Monitor Concept originally used to address computer security. It is our opinion that the traditional concerns of compromise, need-to-know and information flow clearly should be addressed. Information integrity, clearly an important issue in networks qua networks, is not an appropriate topic for modeling at the highest levels of abstraction. Consider the classic computer analogue. The integrity of a message flowing through a network is, in model terms, the integrity of information put into and taken out of an object. In a computer, that translates as the integrity of information written into or read out of a segment. That issue is not addressed inside the model; rather system architecture and design arguments are marshalled to establish that one should accept that reading and writing information satisfies usual integrity requirements. In the network case, at present and with no body of precedents to argue for something stronger, we must accept analogous extrinsic arguments about the integrity of information flowing through the network.[6]

Lastly we consider **denial of service.** Our discussions of networks above do not require that there exists a *liaison* that connects a source subject S_1 with its chosen destination subject S_2. This can result in it being forever impossible for S_1 to transmit data to S_2. This is not generally considered to be a denial of service, because when there is no path between a pair of subjects it is legitimate to assume that the subjects are not permitted to communicate because of either mandatory or discretionary security constraints.

However, when it is only temporarily impossible for the two subjects to communicate because a critical path node between S_1 and S_2 in the network has become unreachable, there is denial of service. Other classes of denial of service include the deliberate or inadvertent misrouting of messages (along paths authorized under the security policy, of course).

While denial of service problems are particularly genant, we do not believe that they can be addressed constructively by formal means at this time, nor are they within the scope of network security standards for state of the art technology on the basis of lack of precedent.

---

[6]Any attempt at modeling integrity will have to provide modeling structure to allow stating of information integrity conditions. This structure will have to contain two levels, one of the information containers (as in traditional models) and one of the information itself that moves between containers. Interestingly, the structure of the SCOMP model suggested a binding of segment objects with segment-status-objects that would allow information integrity conditions to be stated intrinsically within the current modeling framework.

Note

Based on our experience with BLACKER and I-S/A AMPE, we believe that the model of [2], [3], [4], and [6] is perfectly usable for addressing the network security properties that currently can be modeled effectively.

DESIGN

The interplay of the TCB and the model in the classic computer instance needs to be recapitulated for networks. Specifically, the extrinsic design arguments validating and supporting the applicability of modeling concepts is crucial. In discussing networks and, in general, distributed systems, the distributed nature of the TCB (what is the perimeter?); the TCB-to-TCB communication; the multiplicity of network-subject interpretations; and the concept of multilevel *liaisons* must be addressed.

TCBs

It is certainly reasonable to question where security (a. k. a. trust) is in a network context.

I. Suppose all of the access control security were provided by the hosts, rather than any of the security being provided by the network components. Then it would be assumed that each host can be depended upon to guarantee that (1) it is properly cleared to receive data and (2) it will provide the finer degrees of access control that is reflected in the discretionary access control matrix. The clear problem here comes about when a receiving host fails to satisfy these criteria.

II. Suppose the network components are assumed to provide all of the access controls, so that there are no trust properties assumed of the individual hosts. Then while the network can potentially provide access control based on the clearance levels of the individual hosts,[7] it is unclear whether the network can control the finer-grained access controls between processes, service classes, individual users, or combinations of these independent of the hosts' characteristics. Again, it is unclear how well the network can enforce access controls if a host's security or trust attributes are changed and the changes are not reported to the network.

III. It appears that a combination of access control must be provided by both the network and the individual hosts, and it further appears that mutual suspicion (see [7]) must be implemented between all network components. That is, it appears that each network component must be expected to validate its legitimacy to each other network component with which it interacts.

---

[7]By which we mean the range bracketed by the lowest classification of data that can legally be exported from the host and the highest classification of data that can legally be imported by the host.

Agreeing that the TCB is spread out over various network components leaves a minor philosophical question: is the union of the components' TCB's identical to the network TCB? Put another way, is it possible for a component to support a network functionality that is inside the network-TCB perimeter but outside the component-TCB perimeter? The practical resolution is defining network-security-critical as implying component-security-critical and getting on with it.[8]

TCB Communication

As mentioned above, the physical separation of network components reopens the issue of the reliability of communication between subfunctionalities of the network TCB. The conservative course of asserting the need for mutual suspicion is advocated since there is no alternative. Mutual suspicion is clearly required, particularly in the extrinsic arguments brought to bear (arguments that will differ substantially for benign or hostile environments). Consider the following thought-experiment. Suppose components C_1 and C_2 need to communicate, TCB to TCB. If the only means is through compatible dismountable disk packs, C_1 will export information with labels to its disk drive (a la the TCSEC); the disk pack will be physically transported to C_2's disk drive; and C_2 will import the data with labels, also according to the TCSEC. The TCSEC handles that situation directly. What is wrong, we ask, with using the same principles if the disk cables are attached to each other? Should not the import and export rules cover the problem of network components' TCB's communicating over arbitrary communications *liaisons*? (Yes, the nature of the *liaison* will affect the details of the argument; it will not change the points and level of assurance needed.)

The Variety of Network Subjects

It is generally observed in the network security context that a host may not always be the endpoint of a communication between a pair of subjects. The voiced concerns have been that a host can be anything from a personal computer with a modem, to a terminal access controller, to a computer on a local area net not directly connected to the wide-area network, all the way up to one of the preceding on a separate network that is internetted with some other combination of independent networks. Moreover, it is observed that some message traffic is addressed to components of the network itself (e.g., statistical traffic flow data, routing table updates, etc.) and the question is posed as to whether and how such necessary traffic introduces self-referential paradoxes or exceptions to the overall simplified security policy model.

We believe that these topological concerns are not critical at the most abstract level of system specification. Inter-component communication can be viewed as intra-TCB communication using the same mechanism as non-trusted communications, just as is classic kernel communication via IPC. Further, it is not clear that mandatory security enforcement has to be adversely affected by the topology providing the simplified rules are not violated in any point-to-point case. That is, it must be the case that no subject can ever write data on a *liaison* that the *liaison* is not cleared to receive, and reciprocally, that no *liaison* can ever accept data from a subject that the *liaison* is not cleared to receive. Finally, it is required that no subject can ever be permitted to read a *liaison* for which it is not properly cleared.[9] Thus, our simple rules about subjects reading and writing *liaisons* appears to be completely extensible to the macroscopic details of however *liaisons* may have been implemented in terms of switches, repeaters, and so forth.

Multilevel *Liaisons*

There is an opinion that a *liaison* may be either single level or multilevel. The assurance issues relating to multilevel *liaisons* are still under debate; however, it is clear that a multilevel *liaison* needs to provide assurances comparable to those required for multilevel devices in the Criteria. In particular, it must be possible to provide the continuous and unambiguous association between a "trusted label" and the associated data. More importantly, on a mainframe system, a multilevel device may contain a directory (or the directory may be contained elsewhere in the TCB). In either case, the existence and availability of the directory is fundamental to the system's ability to control discretionary accesses between subjects and objects stored on the multilevel device. In the case of single level or multilevel *liaisons*, it appears to be necessary that a similar form of assured control be provided to guarantee that discretionary access controls are assured for messages (which are viewed as data that are "read" from *liaison*-objects).

FORMAL VERIFICATION

Given that network modeling should include both system-wide and component-specific aspects, the question of how to provide design assurance is raised. Clearly providing a formal specification (an FTLS in TCSEC terms) of either a system-level model or a component-level one is straightforward. The harder problem is how to be assured that the collection of component specifications support, are consistent with, and actualize the system-level specification. What is needed is a way to partition a specification for the system into a set of subspecifications whose requirements devolve from the master. This process and its obverse are not easily handled in formal verification systems today. Thus it is inappropriate to include stiff requirements for such partitioning in official criteria. Nevertheless, given uniformity of expression between levels of refinement in a specification language, one can expect careful text-edited transformations and careful inspection to provide adequate assurance of system-to-component conformance until more automated and rigorous means are developed and proven.

---

[8] BLACKER made just this practical decision.

[9] Notice that this is just normal reading and writing restrictions, viewing *liaisons* as objects.

WITHITHER ASSURANCES?

How can this general discussion be related to the assurance requirements of the TCSEC? Tripping fleetingly through the TCSEC's A1 requirements, we conclude that the **system architecture** requirements need to apply completely, albeit with the understanding that the TCB is necessarily distributed. There is no question but that the preceding discussion leads to the conclusion that access control components of the TCB must lie within both the "network" and within certain hosts. (Single-level hosts need not be of concern if the network can guarantee that traffic for which they are not authorized cannot reach them, and can thus be Class D systems. Blacker tends to trust hosts only within their accredited range of security levels. There does not appear to be any particular problem with taking this point of view.) As explained above, the distributed portions of the TCB need to work together not only with least privilege[10] and with least common mechanism, but also with mutual suspicion. All of the requirements for internal structuring are necessary assurance features for a distributed TCB.

The **System Integrity** requirements are of particular interest because of the need to address mutual suspicion between distributed, communicating TCB components. Clearly, in addition to the information security consequences of routing classified data to a network component that may be malfunctioning, there are also clear denial of service concerns that indicate that system integrity should be in the realm of "good networking practice".

Apart from the characteristics of individual network/system components, we see no additional requirement for **covert channel analysis**. One must remember to take into account the effect of the cleartext headers in such components as switches in cases where encryption is involved (the dreaded but misnamed "bypass" problem).

Trusted Facility Management and **Trusted Recovery** also appear to be logical requirements for network security criteria.

Security Testing, as part of the Life-Cycle Assurance requirements, is an omnibus section of the Criteria, since it includes specific reference to correspondence mappings between specification and code. We would observe that much is yet to be learnt about network security testing.[11] However, recognizing that networks are often designed to support frequent configuration changes (more frequent than is true of pure computer systems), it is important to consider whether and how such changes may affect the security of a network. Since each reconfigured component may include a new portion of the distributed network TCB, it would appear that new testing techniques may need to be derived. The mappings from specification to code will also need to be better understood since they will necessarily relate to the replication of several one-or-more-of-a-kind entities.

[10]See also Dorothy Denning's comments on least privilege issues in [5].

[11]But then, one could argue that much is yet to be learnt about testing in general.

Design Specification and Verification assurances have been a major subject matter of this paper. We believe that the foundations of reasonable network security modeling have been established in earlier work and are being extended in the derivation of worked formal specification and verification examples in the work on Blacker and the I-S/A AMPE. The work performed in other exploratory projects, including the Multinet Gateway project will provide additional insight in this area.

Continuous Control, albeit not a specific feature in the Assurance area, is both important and closely related to the **Trusted Path** requirement. As data is transferred throughout the network the distributed TCB must maintain the same continuous control over both its security descriptors[12] as would be required for data in transit in a physical security environment. All security-relevant communications between subjects and TCB components, as well as all communications between distributed TCB components must be unambiguously achievable, and the mutual suspicion principle discussed above needs also to be applied. We again recommend the TCB communication thought-experiment described above.

CONCLUSIONS

Assurance of network security, although it seems to be much different from classical computer security, has in fact proved to be essentially the same. The vital tool in establishing the similarities and highlighting the distinctions is a rigourous application of analogy, comparing the network context to the classical computer security context. This method both confirms the general value of traditional concepts (subjects, objects, security perimeters, formal design verification) and identifies needed research results (network-level data integrity, denial of service, the theory and practice of trusted paths between geographically disparate parts of the TCB). Network security assurance can be done. The precedents lie in the worked examples of classical computer security and in the conservative extensions being constructed in BLACKER and I-S/A AMPE. Yet again there is nothing new under the sun.

ACKNOWLEDGEMENTS

The standardisation effort being undertaken by Sheila Brand was of course the immediate impetus for this description of our views on the network security assurance issues. While we take responsibility for any misstatement

[12]These are the internal data maintained by the TCB that fully characterize the security attributes of each subject and each object in the system. It is a requirement in conventional computer architectures that the TCB be capable of determining the formal security level and access control characteristics of every object and subject. The descriptors in question necessarily include a full characterization of the extent (e.g., size, location, etc.) of the physical representation of each subject or object. The distributed TCB must be capable of deriving the analogous information relative to every subject and object under its control in the network environment.

and possibly blind prejudices herein, we acknowledge the intellectual stimulus of Roger R. Schell, Daniel J. Edwards, Debbie Cooper, Leslee O'Dell, Thomas Parenty, David Solo and Terrance Losonsky. Lastly, although he may not recognize his ideas in this form, we were heavily influenced by James P. Anderson, Technical Conscience of a generation.

REFERENCES

[1] J. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, vol. 1, AFSC/ESD, Hanscom AFB, MA, October, 1972.

[2] D. Bell, "Secure Computer Systems: A Refinement of the Mathematical Model," MTR-2547, vol. 3, The MITRE Corporation, Bedford, MA, 28 December 1973 (also ESD-TR-73-278, vol. 3).

[3] D. Bell and L. La Padula, "Secure Computer Systems: Mathematical Foundations," MTR-2547, vol. 1, The MITRE Corporation, Bedford, MA, 1 March 1973 (also ESD-TR-73-278, vol. 1).

[4] D. Bell and L. La Padula, "Secure Computer Systems: Unified Exposition and Multics Interpretation," MTR-2997, The MITRE Corporation, Bedford, MA, July, 1975.

[5] D. E. Denning, "A Position Statement on Network Security", Proc. DoD Computer Security Center Invitational Workshop on Network Security, DoD Computer Security Evaluation Center, Ft. Meade, MD, 19-21 March 1985.

[6] L. La Padula and D. Bell, "Secure Computer Systems: A Mathematical Model," MTR-2547, vol. 2, The MITRE Corporation, Bedford, MA, November, 1973 (also ESD-TR-73-278, vol. 2).

[7] M. Schroeder, "Cooperation of Mutually Suspicious Subsystems in a Computer Utility," Ph. D. dissertation, M.I.T., Cambridge, MA, 1972 (also MAC-TR-104).

[8] Trusted Computer System Evaluation Criteria, CSC-STD-001-83, DoD Computer Security Center, Ft. Meade, MD, 15 August, 1983.

VERLANGEN: A VERIFICATION LANGUAGE FOR DESIGNS OF SECURE SYSTEMS

Dianne E. Britton
RCA Aerospace and Defense
Advanced Technology Laboratories
Moorestown, NJ

Abstract

Verlangen is a language being developed at RCA for formally specifying and verifying system designs. It supports object-oriented design, concurrency, and levels of refinement, and is appropriate for specifying designs of distributed systems, communication networks, and operating systems, and for verifying that the designs meet security and other kinds of requirements. A compiler translates a Verlangen text, which is a formal specification of a system design and its requirements, directly into a collection of first-order logic definitions and theorems. Proving the theorems verifies that the design satisfies the requirements. The compiler and theorem prover run under the VAX/VMS operating system.

Introduction

Verlangen is a language being developed at RCA for formally verifying that designs of computer systems meet their requirements. ("Verlangen" is a German transitive verb meaning "to require".) It is appropriate for many kinds of systems, including communication networks, distributed systems, and operating systems. To date, Verlangen has been used mostly with designs of secure systems to verify that security requirements are met, but Verlangen's versatility merits its use for systems with other sorts of requirements as well.

The driving force behind the present interest in formal specification and verification of computer system designs is the DoD Trusted Computer Systems Evaluation Criteria [1], which requires verified design at the A1 level. The arguments in favor of verified design rest largely on the desire to increase confidence that a system does or will meet its requirements, and to catch design flaws early in the product life-cycle while the cost of correcting flaws is relatively low. In the case of the Trusted Computer Systems Evaluation Criteria, security requirements are of interest, but the capability to verify designs with other kinds of requirements is equally valuable. Although there are several other languages and systems for specifying and verifying system designs [2,3,4,5,6], RCA is developing Verlangen in belief that it improves upon existing systems, especially with regard to communication networks and distributed systems.

The primary features that make Verlangen a good language for formally specifying and verifying many kinds of systems and requirements are "classes", which support object-oriented design and "levels", which levels of refinement (hierarchical design). Classes and levels decompose both the system design and its verification into tractable units. Verlangen is perhaps unique in supporting the design and verification of truly concurrent systems, such as distributed systems and networks, without mandating a particular model for communications between subsystems. This contrasts Verlangen with, for example, Gypsy [2], which models communication between subsystems by finite buffers with blocking send and receive operations.

A Verlangen text is a formal specification of a system design and its requirements. Many of the features of programming languages that have proved valuable in expressing program specifications are equally valuable for design specifications and are included in Verlangen. These include block structure, identifier scope and visibility rules, user-defined datatypes, and modularity. Like Ada, Modula, Simula and other similar programming languages, Verlangen supports object-oriented design. Nevertheless Verlangen is not a programming language and can be used for specifying designs of systems composed from a combination of hardware and software. An important feature of Verlangen is hierarchical design, which it shares with other design languages [3,4].

A compiler checks a Verlangen text for syntactic correctness and translates it not into executable code but into a collection of definitions, axioms, and theorems in first-order logic. The theorems relate the system requirements to the system design. System requirements are expressed in Verlangen as assertions, which are formulas in first-order logic. Proving the theorems from the definitions, axioms, and rules of inference of first-order logic formally verifies that the design satisfies its requirements. Theorem proving is carried out using a semi-automatic theorem prover.

The desire to verify designs of secure communication networks and distributed systems has played an important role in the development of Verlangen. One such application is a communications network in which all messages in transit over the network are encrypted to prevent them from being tapped. Encryption is also used to ensure that only hosts that are authorized to communicate exchange unencrypted messages. This application, which a simplification of the example described in a previous paper [8], has been specified in Verlangen and verified. Another application is a multi-level secure local area network connecting a number of work stations that are not assumed to deal with or understand security levels. Each work station is assigned a fixed security level and operates at that level only. Interposed between the work stations and the local area network are guards, one for each workstation, which are responsible for enforcing multi-level security by restricting the flow of messages between work stations. The Verlangen specification for this application appears in the Appendix and is used for illustration throughout this paper. A "low water mark" [7] secure system has also been specified and verified using Verlangen.

Object-Oriented Design and Verification

Verlangen supports object-oriented design and verification with a language construct referred to as a "class". The Verlangen concept of class combines the concept of abstract datatype from programming languages with the concept of state machine from specification languages.

Design

A class definition specifies a class of state machines by defining a datatype representing the possible states of the state machines in the class. Classes may be used to design a variety of kinds of system entities which can be modelled as state machines, including data structures, monitors, processes, and systems composed of these.

A specification containing a class definition may instantiate the class one or more times. A class

instantiation, which represents a state machine, is called an object. An object is associated with a sequence of values, called a "history": the first value in the sequence represents the state machine's initial state, and each subsequent value represents each subsequent state. A class definition defines the histories that are possible for objects in the class by specifying an initial value for objects (or a condition on initial values) and some operations or events that yield new values from old ones.

To illustrate the use of the Verlangen class construct, we use the guard unit from the Secure LAN example (Figure 2) as an example. Each guard stands between a work station and the local area network, and runs at a fixed security level. A guard has two functions: labelling with the guard's security level all data going from the guard's work station to the network, and preventing data whose security level is higher than the guard's from reaching the guard's work station.

The Verlangen specification begins by declaring three types:

```
TYPE Lev =
  [unclassified,confidential,secret,topsecret];
TYPE Subj;
TYPE Obj;
```

Values of type Lev represent security levels, values of type Subj identify the individual work stations and their guards, and values of type Obj identify the classified data objects which are generated at the work stations and which may be transmitted over the network. Lev is an enumerated type as in the programming language Pascal.

We model each guard as a state machine and define a Verlangen class called Guard, which specifies a class of state machines that includes all the guard units. Omitting for now the body of the definition (represented by "..."), we write the class definition for Guard as follows:

```
CLASS Guard(CONST max:Lev) IS
  ...
END Guard;
```

This has the effect of declaring Guard to be a datatype, parameterized by a constant max of type Lev.

A guard, viewed as a state machine, performs four different kinds of operations which effect a change of state. These operations are to receive a data object from the network, pass a data object received from the network on to the guard's work station, receive a data object from the work station, and pass a data object received from the work station on to the network. A guard's state consists of components representing the data objects last received from the work station and the network, the destination of the data object last received from the work station, and whether or not the data objects last received have already been passed on (or filtered out). In Verlangen, a state machine's operations (events) are represented by procedures. The definition of Guard defines four procedures, representing the four kinds of operations:

```
CLASS Guard(CONST max:Lev) IS
  VAR netin,netout:Obj;
  VAR dest:Subj;
  VAR netinfull,netoutfull:BOOLEAN INITIALLY FALSE;

  PROCEDURE fromnet(o:Obj; l:Lev) IS ...
  PROCEDURE touser(o:Obj) IS ...
  PROCEDURE fromuser(d:Subj; o:Obj) IS ...
  PROCEDURE tonet(d:Subj; o:Obj; l:Lev) IS ...
  ...

END Guard;
```

The components of the state are represented by the variables declared in the VAR declarations. The variables netin and netout are buffers containing the data object last received from the network and work station, respectively. The variable dest contains the destination work station for the data last received from the work station. The variables netinfull and netoutfull indicate whether or not the netin and netout buffers are full.

The fromnet procedure, which represents the operation of receiving data from the network, illustrates how Verlangen procedures are composed of a procedure header, a precondition, and an effect.

```
PROCEDURE fromnet(o:Obj; l:Lev) IS
  PRECONDITION NOT netinfull;
  EFFECT
    IF l <= max
      THEN (netinfull' & netin' = o)
      ELSE NOT netinfull';
    SAME netoutfull, netout, dest;
END fromnet;
```

The fromnet procedure has two parameters: o is the received data and l is the security level of the work station sending the data.

In a procedure, the formulas appearing between the keywords PRECONDITION and EFFECT are the preconditions for the operation, i.e., the operation occurs in a given state only if the conditions expressed by these formulas are met. In the case of fromnet, there is only one precondition -- that the buffer for holding data objects received from the network be empty.

The formulas appearing after EFFECT specify the relation between the states before and after the operation. Primes refer to values after the operation. SAME specifies that a variable's value remains unchanged by an operation. The effect of the fromnet procedure is to store the data object received from the network in the netin buffer, provided the security level of the sending work station is not greater than the security level of the guard. The variables associated with receiving and sending data objects from the guard's work station remain unchanged by the operation.

Verification

A class definition may include assertions, for which the Verlangen compiler generates verification theorems. Proving the theorems verifies that the class satisfies its assertions. There are two kinds of assertions: "invariants" and "constraints". An invariant is a condition that is expected to be satisfied by every value of every object in the class. A constraint is a condition that is expected to hold between every two subsequent values (states) in every object history.

For example, as a check on the consistency and correctness of the specification, a number of invariants are included in the definition of the Guard class. The filter invariant states that a guard passes an object on to the guard's work station only if the guard previously received the object over the network from a work station with a security level not greater than the guard's security level. The definition for the filter invariant is:

```
INVARIANT filter IS
  FORALL o:Obj
  ( touser(o) =>
      EXISTS g:Guard EXISTS l:Lev
      ( g << THIS &
        g.fromnet(o,l) &
        l <= max )
  );
```

In Verlangen, the "current" value of a variable declared in a class is represented by the value of the variable in THIS state. The "precedes" operator, denoted "<<", relates two states in an object's history. A near-literal reading of the definition for the filter invariant is, "If in THIS state a guard passes a data object o to the guard's work station, then in a preceding state, the guard received object o with a security level l, where l is less than or equal to the guard's security level."

Another invariant, called transport, states that a guard passes a data object onto the network with security level l only if l is the security level of the guard and the guard previously received the data from the guard's work station.

The filter and transport invariants may be regarded as requirements placed on the class. These invariants express properties of the class that are expected to be maintained regardless of the implementation of the class: they refer only to the class's procedures and constant parameters of the class, and not to the class's "internal" variables.

The verification of an invariant is by induction: prove the initial state satisfies the invariant, and prove that if an arbitrary state satisfies the invariant, then the next state does also. Often an invariant is not "inductive", i.e., it is not strong enough for the inductive proof to succeed. Then to obtain a verification, additional invariants, at least one of which is inductive, must be determined and included in the specification. These invariants can then be mentioned in a USING clause associated with a non-inductive invariant to allow the non-inductive invariant to be verified. For example, the filter invariant is not inductive, so a stronger invariant named netinok has been added to the Guard specification and the filter invariant definition has been augmented with a USING clause:

```
INVARIANT netinok IS
  FORALL o:Obj
  ( netinfull & o = netin =>
      EXISTS g:Guard EXISTS l:Lev
      ( g << THIS &
        g.fromnet(o,l) &
        l <= max )
  );

INVARIANT filter USING netinok IS
  FORALL o:Obj
  ( touser(o) => ... );
```

The verification of the Guard class verifies that the netinok invariant is satisfied, and adds netinok to the hypotheses of the verification theorems for the filter invariant: prove that if the netinok invariant holds for all states, then the initial state satisfies the filter invariant, and if the filter invariant holds in an arbitrary state and the netinok invariant holds for all states, then the filter invariant holds for the next state.

As a further check on the consistency and correctness of the specification, two constraints have been added. One of these, the OnlyToUser constraint, insures that only a touser operation empties the netin buffer:

```
CONSTRAINT OnlyToUser IS
  netinfull & NOT netinfull' =>
    touser(netin);
```

As in the EFFECT of a procedure, primes in a constraint definition refer to values after an operation.

Concurrency

Verlangen allows a system to be decomposed into (or composed from, if you prefer) simpler subsystems, each system and subsystem being modelled by a state machine. This approach to system design is generally accepted as being effective for operating systems. For distributed systems and communications network the approach is a natural one: the system naturally decomposes into a set of concurrent, interacting subsystems, which are the host computers, front-ends, gateways, etc. What makes Verlangen especially suitable for distributed systems and communications networks is that true concurrency between subsystems can be specified. Moreover, the communication between subsystems need not be modelled by finite buffers with blocking send and receive operations, which is a point of distinction between Verlangen and Gypsy [2].

In a system composed of a number of concurrent, interacting subsystems, the overall system state is composed of the states of the subsystems, and the overall system changes state in step with the subsystems. This fact can be specified in Verlangen by declaring, in the class definition for the overall system, variables whose datatypes are the classes representing the subsystems.

Returning to the Secure LAN example, the Secure LAN specification defines three classes: WorkStation, Guard, and System. The System class represents the overall system, which is composed of a number of work stations and guards. The work stations and guards appear in the System class definition as variables of type WorkStation and Guard, respectively.

```
CLASS WorkStation(CONST myself:Subj) IS ...

CLASS Guard(CONST max:Lev) IS ...

CLASS System IS
  CONST Clearance(s:Subj):Lev;
  VAR   user(s:Subj):WorkStation(s);
  VAR   guard(s:Subj):Guard(Clearance(s));
  ...
 END System;
```

Communication, or synchronization, among concurrent subsystems is specified in Verlangen by SYNC statements, which correllate events occurring in the subsystems. A SYNC statement says that two or more events, occurring in different subsystems, are manisfestations of a single event in the overall system. The System class definition of the Secure LAN specification includes SYNC statements which state how the work stations interact with the guards, and how the guards interact with each other. The following SYNC statement specifies that a work station's act of sending an object corresponds to the guard's act of receiving the object from the work station:

```
FORALL s,d:Subj FORALL o:Obj
  SYNC user(s).send(d,o),
       guard(s).fromuser(d,o);
```

Two other SYNC statements specify that receiving an object at a work station corresponds to sending the object at the work station's guard, and that a guard receives an object from the network if and only if some

other guard "simultaneously" sends the object over the network.

To check the consistency of the specification of a system composed of interacting subsystems, assertions relating the subsystem states to the overall system state may be specified and verified. For the Secure LAN example, an invariant, called Origination, is included in the System class definition which asserts that any object known at a user work station was created by some user work station on the network. This rules out, for example, a design in which the guards spontaneously create objects of their own.

```
INVARIANT Origination
      USING WorkStation.knowledge,
            Guard.filter,
            Guard.transport
IS FORALL s:Subj FORALL o:Obj
   ( user(s).knows(o) =>
       EXISTS sys:LANSystem
       ( sys << THIS &
         sys.user(Originator(o)).write(o)
       )
   );
```

The invariants mentioned in the USING clause above are invariants for the WorkStation and Guard subsystems. This illustrates how Verlangen allows the verification of the overall system to be decomposed by treating the invariants and constraints of the subsystems as lemmas which are separately verified.

Hierarchical Design

Verlangen supports design by successive levels of refinement, sometimes called "hierarchical design". A Verlangen specification may be written as one or more levels, each level being a complete specification of the whole system. The levels are totally ordered, the first level presenting the most abstract view of the system and each successive level presenting a more concrete specification than the preceding one. Each successive level includes a mapping that specifies how it relates to its predecessor. Using terminology generally accepted for hierarchical design, we call the first and last levels "top" and "bottom", respectively, and all levels except the first we refer to as "lower".

Typically, a Verlangen specification consists of only one or two levels, but occasionally the need for more than two levels arises. Verifying a specification consisting of more than one level shows that the individual levels are self-consistent and that neighboring levels are consistent with each other.

A two-level specification is usually organized so that the top level represents a set of requirements placed on the system, and the bottom level represents the system design. For example, to verify that a system design meets security requirements, a good approach is to specify the security requirements at the top level and the system design at the bottom level. This approach allows a set of requirements, e.g., a model of multilevel security, to be used over and over again with different system designs.

The Secure LAN example is a two-level Verlangen specification, where the top level specifies a model of multilevel security and the bottom level specifies the design of the secure local area network. The Verlangen fragments given earlier in this paper are drawn from the bottom level of the Secure LAN specification. This section presents the top level of the Secure LAN specification, the mapping between the top and bottom levels, and the method used to show consistency between the levels.

An Example Two-Level Specification

The top level of the Secure LAN example is an abstract model for multilevel security; the specification names this level mls. The next lower level, which in this case is the bottom level, is the design of a secure local area network distributed system; this level is named lan. The overall organization of the Secure LAN specification is as follows:

```
LEVEL mls IS
  ...
END mls;

LEVEL lan REFINES mls IS
  ...
END lan;
```

Like the Bell-LaPadula model of multilevel security [9], the mls level is based on subjects, objects, and a partially-ordered set of security levels. The following declarations appear in level mls:

```
TYPE Subj;
TYPE Obj;
TYPE Lev;
CONST Dom(11,12:Lev):BOOLEAN SATISFIES
  FORALL 1:Lev Dom(1,1) &
  FORALL 11,12:Lev
  ( Dom(11,12) & Dom(12,11) => 11=12 ) &
  FORALL 11,12,13:Lev
  ( Dom(11,12) & Dom(12,13) => Dom(11,13) );
```

The SATISFIES clause in the declaration for Dom states that the relation is reflexive, antisymmetric, and transitive. Thus, Dom imposes a partial ordering on Lev values.

Level mls defines a single class called System, which represents the overall system. The system state is represented by the constants and variables declared in the System class definition: a constant function called Clearance gives the security level associated with each subject, a variable function called Classification gives the security level (currently) associated with each object, and two boolean functions called ReadAcc and WriteAcc determine whether a subject can read or write an object. At this level of the specification, work stations and guards are not represented.

```
CLASS System IS
  CONST Clearance(s:Subj):Lev;
  VAR   Classification(o:Obj):Lev;
  VAR   ReadAcc(s:Subj; o:Obj):BOOLEAN;
  VAR   WriteAcc(s:Subj; o:Obj):BOOLEAN;
  ...

END System;
```

Level mls does not specify any events for the System class. It does, however, specify two invariants, a constraint, and initial conditions.

The SimpleSecurityCondition invariant specifies that a subject has read access to an object only if the security level of the subject dominates the security level of the object.

```
INVARIANT SimpleSecurityCondition IS
  FORALL s:Subj FORALL o:Obj
  ( ReadAcc(s,o) =>
      Dom(Clearance(s),Classification(o)) );
```

Similarly, the StarProperty invariant states that a subject has write access to an object only if the security level of the subject is dominated by the security level of the object.

```
INVARIANT StarProperty IS
  FORALL s:Subj FORALL o:Obj
  ( WriteAcc(s,o) =>
      Dom(Classification(o),Clearance(s)) );
```

The NoDowngrading constraint states that the classification of an object cannot be decreased by any event.

```
CONSTRAINT NoDowngrading IS
  FORALL o:Obj
    Dom(Classification'(o),Classification(o));
```

These three assertions together are meant to express the requirement that information does not flow downward, i.e., from a higher security level to a lower one.

The initial conditions simply restate the SimpleSecurityCondition and StarProperty invariants, requiring only that the invariants be satisfied in the initial state.

Since there are no events specified, verification for the mls level consists of proving that the initial conditions (taken together) imply the invariants, which is trivial in this case. Verification that level lan is consistent with level mls consists of proving that the initial conditions, invariants, and constraint of level mls, and the SATISFIES clause of Dom, are satisfied by level lan and the mapping between the two levels.

Mapping Between Levels

To verify that two neighboring levels are consistent with each other, there must be a mapping that determines how the entities in the upper level are represented in the lower level. In Verlangen, there is an implicit mapping between identifiers with the same name in neighboring levels. In the Secure LAN specification, for example, there is an implicit mapping for the identifiers Subj, Obj, and Lev, each of which is declared both in level mls and in level lan. Likewise, there is an implicit mapping for the identifier Clearance, declared both in level mls in class System and in level lan in class LANSystem, which refines System.

The Verlangen compiler checks that declarations of the same identifier in neighboring levels are compatible. Compatible declarations need not be identical. For example, the declarations for Lev in levels mls and lan are compatible, though not identical; the declaration in level lan is merely more specific than the one in mls. Level mls declares Lev to be an arbitrary type, giving the type the name Lev to distinguish it from other types:

```
TYPE Lev;
```

Level lan declares Lev to be an enumerated type (as in the programming language Pascal) with values unclassified, confidential, secret, and topsecret:

```
TYPE Lev =
  [unclassified,confidential,secret,topsecret];
```

The implicit mapping associated with identical identifiers is insufficient to describe the mapping between neighboring levels. A mapping statement, which can appear only in a lower level, provides an explicit mapping from a constant or variable declared declared in the next higher level to its representation in the lower level. For example, level lan gives the following mapping for the constant Dom:

```
MAP Dom(11,12) INTO (12 <= 11);
```

This mapping statement says that the relation Dom, declared in level mls, is represented in level lan by the relation "<=" ("less than or equal to"). Additional mapping statements appear in the System class definition:

```
MAP Classification(o) INTO Clearance(Originator(o));
MAP ReadAcc(s,o) INTO user(s).knows(o);
MAP WriteAcc(s,o) INTO Originator(o) = s;
```

Verlangen provides two more constructs for specifying the mapping between levels. These are the REFINES clause of a class declaration and the REFINES clause of a procedure declaration.

A class at one level may be represented by one or more classes at the next level. Mappings between classes are specified by including a REFINES clause in the lower-level class definitions. For example, a class intended to represent an arbitrary host computer in a distributed system might be represented at the next level by several different classes, each representing a different kind of host.

```
LEVEL upper IS
  ...
  CLASS Host ...
  ...
END upper;

LEVEL lower REFINES upper IS
  ...
  CLASS FileServer REFINES Host ...
  CLASS UserHost REFINES Host ...
  CLASS GateWay REFINES Host ...
  ...
END lower;
```

Similarly, an event defined in an upper-level class may be represented by more than one event in a representation of the class at the next lower level. For example, consider a class representing a file server with an event named Alter:

```
CLASS FileServer IS
  ...
  PROCEDURE Alter ...
  ...
END FileServer;
```

At the next lower level, the Alter event may be represented by two different events:

```
CLASS FileServer2 REFINES FileServer IS
  ...
  PROCEDURE Rewrite REFINES Alter ...
  PROCEDURE Append REFINES Alter ...
  ...
END FileServer2;
```

Consistency Between Levels

Verifying the consistency of successive refinements with each other entails verifying that the lower-level representation of each upper-level entity is consistent with its upper-level specification. The theorems generated to verify inter-level consistency are called "mapping theorems".

A lower-level constant is shown to be consistent with an upper-level constant by proving that the lower-level

representation satisfies the upper-level constant's SATISFIES clause, if there is one.

A lower-level class is shown to be consistent with an upper-level class by proving that it maintains the invariants and constraints of the upper-level class. So, for each upper-level invariant, a mapping theorem is produced which states that the lower-level initial conditions and the mapping between the levels together imply the upper-level initial conditions. The rest of the mapping theorems produced for a class refinement fall into three categories.

The first category contains mapping theorems for each lower-level procedure that REFINES an upper-level procedure. These mapping theorems state that for every occurrence of the lower-level event, there is an occurrence of the corresponding upper-level event.

The second category contains mapping theorems for procedures that do not REFINE upper-level procedures. These mapping theorems state that for every upper-level invariant, if a lower-level state satisfies the upper-level invariant and an event defined by the procedure occurs, then the next state also satisfies the upper-level invariant.

The third category contains mapping theorems for lower-level classes which contain no procedures. In this case, the mapping theorems state that the upper-level invariants are provable directly from the lower-level class invariants class (and the mappings).

For example, class LANSystem REFINEs class System. Since LANSystem contains no procedures, the System invariants must be provable directly from the LANSystem invariants. To make the System SimpleSecurityCondition invariant provable from the LANSystem invariants, another invariant -- also called SimpleSecurityCondition -- has been added to LANSystem:

```
INVARIANT SimpleSecurityCondition
      USING WorkStation.knowledge,
            Guard.filter,
            Guard.transport
IS FORALL s:Subj FORALL o:Obj
   ( user(s).knows(o) =>
       Clearance(Originator(o)) <= Clearance(s) );
```

Translation to First-Order Logic

This section describes the translation from a Verlangen specification to typed first-order logic with equality. The translation is illustrated below using a notation for first-order logic which includes declarations for types, functions, axioms, and theorems. Formulas may be quantified existentially with "exists" or universally with "for". The operators "NOT", "&", "v", "->", "<=>" are used for negation, conjunction, disjunction, implication, and equivalence, respectively. The symbols "=", "<>", "<", "<=", ">", ">=" are the relational operators having their common interpretation.

General

For each class, the Verlangen compiler generates two files of first-order logic "code". The "definitions" file contains the declarations and definitions of types, constants, functions, and axioms that comprise the definition of the class expressed in first-order logic. The "theorems" file contains the theorems for verifying that the class is self-consistent.

For each class defined as a refinement of another class, the compiler also generates a "mappings" file, which contains the mapping definitions and the mapping theorems for verifying that the class is a consistent refinement of its parent class. The mapping file obtains additional definitions and declarations from the "definitions" files for the class and its parent.

Structuring the first-order logic code in this manner allows the verification to be carried out separately for each class and for the mappings between classes.

Classes

A class definition specifies a class of state machines. The first-order logic translation of a class definition declares a type, which has the same name as the class and whose values represent the states of the state machines in the class. A relation precedes (corresponding to the "<<" relation) and a function next is defined for values of this type. The precedes relation defines a partial ordering on values of the type, while next is a successor function satisfying precedes(x,next(x)). When given a state (in the history of a state machine), next yields the next state in the history. For example, the class definition of Guard generates the following first-order logic definitions:

```
TYPE Guard

FUNCTION precedes: Guard X Guard -> BOOLEAN
DEFINE precedes(g1,g2) BY
          g1 = g2 v
          exists g3:Guard
          ( precedes(g1,g3) & precedes(g3,g2) )
AXIOM for g1,g2:Guard
      ( precedes(g1,g2) & precedes(g2,g1) => g1 = g2 )

FUNCTION next: Guard -> Guard
AXIOM for g:Guard
        precedes(g,next(g))
```

To distinguish functions, such as next and guard, defined for one class from those defined for another, the translation produced by the compiler includes the name of the defining class in the names of the functions. For example, the next function for the Guard class is really named Guard-next. For readability, however, the class name is omitted from the examples presented here.

Constants and Variables. Constants and variables declared within a class definition are translated into state functions. For constants, an axiom states that the value of a constant remains unchanged from one state to the next. For example, the declaration of the constant max in the Guard class translates into first-order logic as follows:

```
FUNCTION max: Guard -> Lev
AXIOM for g:Guard
        max(next(g)) = max(g)
```

For class-type variables, an axiom states that the value of the variable either remains unchanged from one state to the next or is the next value in the history of the variable. For example, the declaration of the variable user in the LANSystem class translates into:

```
FUNCTION user: LANSystem X Subj -> WorkStation
AXIOM for this:LANSYSTEM for s:Subj
      ( user(next(this),s) = user(this,s) v
        user(next(this),s) = next(user(this,s)) )
AXIOM for this:LANSystem for s:Subj
        myid(user(this,s)) = s
```

Initial Condition. An initial condition, which constrains the initial state or set of possible initial states for a class of state machines, is specified in Verlangen by INITIALLY clauses attached to VAR

declarations, by IS clauses attached to CONST declarations, and by INITIALLY statements. The first-order logic translation for a class represents the initial condition by a predicate, i.e., a boolean function, which determines whether or not a state is a possible initial state. The initial condition for the Guard class is defined as follows:

```
FUNCTION initial: Guard -> BOOLEAN
DEFINE initial(g) BY
          for g2:Guard NOT precedes(g2,g) &
          NOT netinfull(g) & NOT netoutfull(g)
```

Procedures. A procedure represents an event (or operation), and translates into a predicate of one or more arguments, the first argument corresponding to a state (in which the event occurs) and the remaining arguments parameterizing the event. The predicate defines a relation between the state represented by the first argument and the next state.

A procedure has two components, called the "precondition" and the "effect". The precondition states the condition(s) which must be satisfied by a state in order for the corresponding event to occur in that state. The effect states the relation between the state in which the event occurs and the next state. The predicate representing the event is defined by the conjunction of the precondition and the effect. The following is the first-order logic translation for the fromnet procedure in the Guard class:

```
FUNCTION fromnet: Guard X Obj X Lev -> BOOLEAN
DEFINE fromnet(g,o,1) BY
          NOT netinfull(g) &
          ( 1 <= max(g) =>
              netinfull(next(g)) &
              netin(next(g)) = o ) &
          ( 1 > max(g) =>
              NOT netinfull(next(g))) &
          netoutfull(next(g)) = netoutfull(g) &
          netout(next(g)) = netout(g) &
          dest(next(g)) = dest(g)
```

Event Synchronization. The SYNC statement correlates events occurring in two or more different subsystems. The first-order logic translation is an AXIOM stating, for all states of the overall system of which the subsystems are a part, the equivalence of the predicates representing subsystem events. For example, the first SYNC statement in the LANSystem class, i.e.,

```
FORALL s,d:Subj FORALL o:Obj
  SYNC user(s).send(d,o),
       guard(s).fromuser(d,o);
```

translates into

```
AXIOM for x:LANSystem for s,d:Subj for o:Obj
      ( send(user(x,s),d,o) <=>
          fromuser(guard(x,s),d,o) )
```

Invariants. An invariant is a condition that must be satisfied by every state of every state machine in the class. An invariant translates into a predicate, whose single argument is a state. The filter invariant in the Guard class translates into:

```
FUNCTION filter: Guard -> BOOLEAN
DEFINE filter(s) BY
          for o:Obj
            ( touser(s,o) =>
                exists g:Guard exists 1:Lev
                ( precedes(g,s) &
                  fromnet(g,o,1) &
                  1 <= max(s) ))
```

To verify that a class satisfies an invariant, it is necessary to prove two kinds of theorems: an initial condition theorem and one or more "induction" theorems stating that if an invariant holds in a given state then the invariant also holds in the next state.

An initial condition theorem states that the invariant is satisfied in the initial state. That is, if a state satisfies the initial condition for the class, then the state also satisfies the invariant. The initial condition theorem for the knowledge invariant in the WorkStation class is:

```
THEOREM for w:WorkStation
          ( initial(w) => knowledge(w) )
```

Induction theorems insure that an invariant remains true for the rest of the states in a state machine's history. SYNC and PROCEDURE statements cannot both appear within a class definition and the induction theorems take on different forms depending on whether the class is defined using SYNC or PROCEDURE statements.

When a class is defined using PROCEDURE statements, there is an induction theorem for every invariant-procedure pair. This theorem states that if the invariant is satisfied in some state, and if an event represented by the procedure occurs in that state, then the next state also satisfies the invariant. For example, the induction theorem stating that the receive procedure in the WorkStation class maintains the knowledge invariant is:

```
THEOREM for w:WorkStation for o:Obj
          ( knowledge(w) &
            receive(w,o) => knowledge(next(w)) )
```

When the definition of an invariant contains a USING clause, the hypothesis of the theorems produced for the invariant is augmented by the assertions mentioned in the USING clause. For For example, the theorem stating that the fromnet procedure in the Guard class maintains the filter invariant is:

```
THEOREM for g:Guard for o:Obj for 1:Lev
          ( filter(g) &
            for g2:Guard netinok(g2) &
            fromnet(g,o,1) => filter(next(g)) )
```

When a class is defined using SYNC statements, for each invariant there is an induction theorem stating that if the invariant is satisfied in some state, then the invariant is also satisfied in the next state. For example, the following induction theorem is generated for the Origination invariant in class LANSystem.

```
THEOREM for this:LANSystem
          ( Origination(this) &
            for w:WorkStation knowledge(w) &
            for g:Guard filter(g) &
            for g:Guard transport(g)
              => Origination(next(this)) )
```

For each class-type variable x declared in class C to be of type T, the following axiom is provided:

```
AXIOM for this:C for y:T
        ( precedes(y,x(this)) =>
            exists prior:C
            ( precedes(prior,this) &
              y = x(prior) ))
```

This axiom really is redundant, since the following invariant can always be verified without recourse to the axiom:

```
INVARIANT xInvariant IS
  for y:T
    ( y << x =>
        exists prior:C
        ( prior << THIS & y = prior.x ))
```

The redundant axiom is provided because the corresponding invariant is so frequently needed in the proofs of other invariants.

Constraints. A constraint places a condition on how any two successive states, i.e., the states before and after an event, may relate to each other. A constraint translates into a predicate, whose single argument is a state. The OnlyToUser constraint in the Guard class translates into:

```
FUNCTION OnlyToUser: Guard -> BOOLEAN
DEFINE OnlyToUser(g) BY
          netinfull(g) &
          NOT netinfull(next(g))
            => touser(g,netin(g))
```

To verify that a class satisfies a constraint, it is necessary to prove a constraint theorem for each kind of event. A constraint theorem insures that the states before and after the event in question satisfy the constraint. The theorem stating that the fromnet procedure of the Guard class satisfies the OnlyToUser constraint is:

```
THEOREM for g:Guard for o:Obj for l:Lev
        ( fromnet(g,o,l) => OnlyToUser(g) )
```

Levels and Mappings

This section describes the first-order logic representations of mappings for classes, variables, constants, and the theorems for verifying the consistency between levels.

Classes When one class refines another, the classes are represented by two different datatypes. Mapping one class to the other entails making the two types equal. This is accomplished by declaring, in the mappings file for the refining class, that the refined class is a subtype of the refining class (so that every value in the refined class is in the refining class) and including an axiom stating that every element that is in the refining class is in the refined class. For example, if class C2 refines class C1, the mappings file for C1 includes the following:

```
INCLUDE C1.def "C1.def is the definitions file for C1"
INCLUDE C2.def "C2.def is the definitions file for C2"
TYPE C1 OF C2
AXIOM for c2:C2
        exists c1:C1 ( c1 = c2 )
```

This approach allows a level to define several different refinements of a class, while keeping the type names associated with the different refinements separate.

Variables. Whenever a specification provides a lower-level representation of an upper-level variable, through redeclaration or a MAP statement, an axiom in the mappings file for the lower-level class makes the requisite mapping. For example, the mappings file for LANSystem includes the following axiom for ReadAcc:

```
AXIOM for sys:System for s:Subj for o:Obj
        System__ReadAcc(sys,s,o) =
          WorkStation__knows(LANSystem__user(sys,s),o)
```

Constants. Mappings for constants are treated similar to the way in which mappings for variables are treated, except that the translation for a mapping of a constant declared with a SATISFIES clause includes a theorem stating that the lower-level representation satisfies the condition expressed in the SATISFIES clause. Consider the constant function Dom, declared in level mls of the Secure LAN specification. The translation for the Dom declaration includes a predicate, named Dom-Axiom, which expresses Dom's SATISFIES clause.

```
FUNCTION Dom_Axiom: -> BOOLEAN
DEFINE Dom_Axiom BY
        for 1:Lev Dom(1,1) &
        for 11,12:Lev
        ( Dom(11,12) & Dom(12,11) => 11=12 ) &
        for 11,12,13:Lev
        ( Dom(11,12) & Dom(12,13) => Dom(11,13) )
```

A MAP statement in level lan maps Dom into the relation "<=" defined on values of type Lev. The mappings file for level lan defines Dom, which is declared but not defined in the definitions file for level mls, and includes a theorem stating that Dom-Axiom is satisfied by the mapping.

```
DEFINE Dom(11,12) BY 12 <= 11
THEOREM Dom_Axiom
```

Consistency Between Levels. Whenever a lower-level class refines an upper-level class, the lower-level class must be shown to satisfy the upper-level assertions. For each invariant, there is a mapping theorem stating that the lower-level initial conditions imply the upper-level initial conditions. The proof of the initial condition mapping theorem utilizes the mappings for the upper-level variables and constants. The initial condition mapping theorem for LANSystem, which refines System is:

```
THEOREM for sys:System
        ( initial__LANSystem(sys) =>
            initial__System(sys) )
```

For each event in a class refinement, there are mapping theorems for verifying that the event maintains the upper-level invariants and constraints.

For every event that refines an upper-level event, an event mapping theorem is produced which verifies that the event is consistent with the event it is refining. Informally, the theorem states, "If the lower-level event occurs in a given state, then the upper-level event can be shown to occur in the same state." Proving the mapping theorem implies that the lower-level event satisfies the upper-level invariants and constraints satisfied by the upper-level event. For example, consider the following fragment of a Verlangen specification

```
CLASS FileSystem IS
  ...
  INVARIANT StarProperty ...
  PROCEDURE Alter(f:File) ...
  ...
END FileSystem;
...
CLASS FileSystem2 REFINES FileSystem IS
  ...
  PROCEDURE Append(f:File;d:Data) REFINES Alter ...
  PROCEDURE Read(f:File;d:Data) ...
  ...
END FileSystem2;
```

The following event mapping theorem is produced for Append, since it refines the upper-level procedure Alter:

```
THEOREM for s:FileSystem
        ( exist f:File exist d:Data append(f,d) =>
            exist f:File alter(f) )
```

If an event in a class refinement is a new event introduced in the refinement, induction mapping theorems are produced for the upper-level invariants and constraint theorems are produced for the upper-level constraints. For example, the following induction mapping theorem is produced for the procedure Read, above.

```
THEOREM for sys:FileSystem
          for f:File for d:Data
          ( StarProperty(sys) &
            Read(sys,f,d) => StarProperty(next(sys)) )
```

If a class defines no events, i.e., contains no PROCEDURE definitions, it must still be shown that the class satisfies all upper-level invariants and constraints. The following theorem is the induction mapping theorem for the upper-level SimpleSecurityCondition invariant in the Secure LAN specification.

```
THEOREM for sys:System
  ( System_SimpleSecurityCondition(sys) &
    for lsys:LANSystem
      LANSystem_SimpleSecurityCondition(lsys) &
    for lsys:LANSystem
      LANSystem_Origination(lsys)
    => System_SimpleSecurityCondition(next(sys)) )
```

The following is the constraint mapping theorem for the NoDowngrading constraint.

```
THEOREM for sys:System
          System_NoDowngrading(sys)
```

Operational System

Verlangen is supported by a compiler and a theorem prover, both of which run under the VAX/VMS operating system. The compiler is being developed at RCA. Currently the compiler has been implemented for a subset of the language, and is expected to be extended to the entire language before the end of 1985. The theorem prover is part of the Verus verification system [6], a product of Gould Corporation.

To carry out a specification and verification using Verlangen, the user first writes a Verlangen specification and then passes it to the Verlangen compiler. The compiler does extensive syntactic checks on the specification and reports any errors. When the specification is error-free, the compiler translates the specification into a collection of declarations, definitions, axioms, and theorems, all expressed in the Verus language. The statement of each theorem references a separate proof outline file. The user creates the proof outline files, leaving them empty or filling them in as necessary, and then submits the theorems to the Verus verification system for proof.

The Verus language is a notation for first-order logic with types and equality. A Verus specification may include declarations of types, constants, and functions, as well as theorems and directives to the theorem prover. Types INTEGER and BOOLEAN are built in, and Presburger arithmetic is supported. A parser supplied with the Verus verification system translates specifications written in the Verus language to an intermediate form acceptable to the Verus theorem prover.

The Verus theorem prover uses the Hintikka tableau method. The basic strategy of this method is to construct a tree of formulas, which begins as a single branch consisting of formulas known to be true and the negation of the theorem to be proved. Each formula, including the negation of the theorem, is reduced to one or more simpler formulas which are consequences of it, and these formulas are added to the branch containing the formula. Universally and existentially quantified formulas are simplified by instantiating them appropriately. A disjunction is reduced by splitting the branch containing the disjunction into several new branches, each new branch headed by one of the disjuncts. A branch stops growing when its last-computed formula contradicts some formula appearing on the path from the end of the branch back to the root of the tree. The proof is complete when all branches have stopped growing.

The Verus implementation of this basic method allows a proof outline to accompany any theorem. A proof outline may include formulas known or assumed to be true, directives to instantiate universally and existentially quantified formulas, theorems to be proved in the context of the theorem currently being proved, and directives to block and unblock function definitions. The proof outline provides a means for naturally structuring a proof, and is sometimes necessary to prove the theorem in a reasonable amount of time and space.

Summary

Verlangen is a language being developed at RCA for formally specifying and verifying system designs. A Verlangen text, which is a specification of a system design and its requirements, can be formally verified, through mathematical proof, to show that the design satisfies the requirements. Verlangen supports and encourages the design of a system and its verification to be decomposed into simple, tractable units.

Verlangen supports object-oriented design and levels of refinements. It also supports true concurrency without mandating a particular model for communications between subsystems. In appearance, Verlangen resembles a high-level programming language having types, block structure, identifier scope and visibility rules, modules, etc. System behavior and system requirements are specified through definitions and assertions expressed by formulas in first-order logic. The logic component of Verlangen is typed, with equality, and with universal and existential quantification.

Verlangen is supported by a compiler and an automatic theorem prover. A compiler checks a Verlangen text for syntactic correctness and translates it into a collection of definitions, axioms, and theorems in first-order logic. Proving the theorems verifies that a specification is consistent, and in particular that the system requirements are satisfied by the system design. The theorems are proved by submitting them to a theorem prover, which may require a proof outline to guide the search for a proof.

Verlangen can be used to specify and verify systems composed of hardware or software elements, or a combination of both. Verlangen is especially suitable for systems composed of several different elements operating concurrently, e.g., a distributed computer network, and for systems required to satisfy multilevel security requirements. Verlangen is intended to satisfy the "verified design" requirement imposed on multilevel secure computer systems at the A1 level of the secure computer systems evaluation criteria set forth by the Department of Defense.

Because Verlangen system specifications are verifiable, Verlangen satisfies the need to have confidence that a

system design meets its requirements before costly time and effort is invested in system implementation. In addition, the formality of the specification language serves to add rigor to the specification process, to achieve unambiguous and complete specifications.

Acknowledgements

Verlangen is being developed through the IR&D program at RCA Aerospace and Defense. The author wishes to acknowledge the past and continuing support of RCA Government Communications Systems and RCA Advanced Technology Laboratories. Special thanks to Beth Reynolds and Myles Boddy for carrying out several verifications using Verlangen even while the language and compiler were still under development, and to Frederick Druseikis for his thoughtful criticism of earlier drafts of this paper.

References

[1] Department of Defense Computer Security Center, DoD Trusted Computer System Evaluation Criteria, CSC-STD-001-83, Aug 1983.

[2] D.I. Good, R.M. Cohen, C.H. Hoch, L.W. Hunter, and D.F. Hare, Report on the Language Gypsy Version 2.0, ICSCA-CMP-10, The University of Texas at Austin, Rev Sep 1978.

[3] K.N. Levitt, L. Robinson, and B.A. Silverberg, The HDM Handbook, Vols 1-3, Computer Science Lab., SRI International, Menlo Park, CA, June 1979.

[4] R. Kemmerer, "FDM -- A Specification and Verification Methodology", Proceedings of the Third Seminar on the DoD Computer Security Initiative Program, National Bureau of Standards, Gaithersburg, MD, Nov 1980.

[5] R.R. Musser, "Abstract Data Type Specification in the AFFIRM System", IEEE Trans. on Software Engineering, SE-6,1 Jan 1980.

[6] D. Craigen, "Ottawa Euclid and EVES: A Status Report", Proceedings of the 1984 Symposium on Security and Privacy, Oakland, CA, May 1984.

[7] M.H. Cheheyl, M. Gasser, G.A. Huff, J.K. Millen, "Verifying Security", Computing Surveys, Vol 13 No 3, Sep 1981.

[8] D.E. Britton, "Formal Verification of a Secure Network with End-to-End Encryption", Proceedings of the 1984 Symposium on Security and Privacy, Oakland, CA, May 1984.

[9] D.E. Bell and L.J. LaPadula, Secure Computer System: Unified Exposition and Multics Interpretation, ESD-TR-75-306, Mitre Corporation, Bedford, MA, Mar 1976.

[10] Verus User Documents, Compion Corporation, a subsidiary of Gould Inc., 1984.

Appendix: Complete Secure LAN Specification

```
LEVEL mls IS

  TYPE Lev;
  CONST Dom(l1,l2:Lev):BOOLEAN SATISFIES
    FORALL l:Lev Dom(l,l) &
    FORALL l1,l2:Lev
    ( Dom(l1,l2) & Dom(l2,l1) => l1=l2 ) &
    FORALL l1,l2,l3:Lev
    ( Dom(l1,l2) & Dom(l2,l3) => Dom(l1,l3) );

  TYPE Subj;
  TYPE Obj;

  CLASS System IS
    CONST Clearance(s:Subj):Lev;
    VAR   Classification(o:Obj):Lev;
    VAR   ReadAcc(s:Subj; o:Obj):BOOLEAN;
    VAR   WriteAcc(s:Subj; o:Obj):BOOLEAN;

    INITIALLY
      FORALL s:Subj FORALL o:Obj
      ( ReadAcc(s,o) =>
          Dom(Clearance(s),Classification(o)) );

    INITIALLY
      FORALL s:Subj FORALL o:Obj
      ( WriteAcc(s,o) =>
          Dom(Classification(o),Clearance(s)) );

    INVARIANT SimpleSecurityCondition IS
      FORALL s:Subj FORALL o:Obj
      ( ReadAcc(s,o) =>
          Dom(Clearance(s),Classification(o)) );

    INVARIANT StarProperty IS
      FORALL s:Subj FORALL o:Obj
      ( WriteAcc(s,o) =>
          Dom(Classification(o),Clearance(s)) );

    CONSTRAINT NoDowngrading IS
      FORALL o:Obj
        Dom(Classification'(o),Classification(o));
  END System;

END mls;

LEVEL lan REFINES mls IS

  TYPE Lev =
    [unclassified,confidential,secret,topsecret];
  MAP Dom(l1,l2) INTO (l2 <= l1);

  TYPE Subj;
  TYPE Obj;
  CONST Originator(o:Obj):Subj;

  CLASS WorkStation(CONST myself:Subj) IS
    VAR knows(o:Obj):BOOLEAN
          INITIALLY FALSE;

    PROCEDURE read(o:Obj) IS
      PRECONDITION knows(o);
      EFFECT FORALL o2:Obj
               SAME knows(o2);
    END read;
```

```
  PROCEDURE write(o:Obj) IS
    PRECONDITION Originator(o) = myself;
                 NOT knows(o);
    EFFECT FORALL o2:Obj
             SAME knows(o2)
             EXCEPT o2 = o -> TRUE;
  END write;

  PROCEDURE send(r:Subj; o:Obj) IS
    PRECONDITION knows(o);
    EFFECT FORALL o2:Obj
             SAME knows(o2);
  END send;

  PROCEDURE receive(o:Obj) IS
    EFFECT FORALL o2:Obj
             SAME knows(o2)
             EXCEPT o2 = o -> TRUE;
  END receive;

  INVARIANT knowledge IS
    FORALL o:Obj
    ( knows(o) =>
        EXISTS w:WorkStation
        ( w << THIS &
          ( w.receive(o) v w.write(o) )
        )
    );
END WorkStation;

CLASS Guard(CONST max:Lev) IS
  VAR netin,netout:Obj;
  VAR netinfull:BOOLEAN INITIALLY FALSE;
  VAR netoutfull:BOOLEAN INITIALLY FALSE;
  VAR dest:Subj;

  PROCEDURE fromnet(o:Obj; l:Lev) IS
    PRECONDITION NOT netinfull;
    EFFECT
      l <= max => netinfull' & netin' = o;
      l > max => NOT netinfull';
      SAME netoutfull,netout,dest;
  END fromnet;

  PROCEDURE touser(o:Obj) IS
    PRECONDITION netinfull;
                 o = netin;
    EFFECT NOT netinfull';
           SAME netoutfull,netout,dest;
  END touser;

  PROCEDURE fromuser(d:Subj;
                     o:Obj) IS
    PRECONDITION NOT netoutfull;
    EFFECT
      SAME netin,netinfull;
      netoutfull';
      dest' = d;
      netout' = o;
  END fromuser;

  PROCEDURE tonet(d:Subj;
                  o:Obj; l:Lev) IS
    PRECONDITION netoutfull;
                 d = dest;
                 o = netout;
                 l = max;
    EFFECT SAME netin,netinfull;
           NOT netoutfull';
  END tonet;

  INVARIANT netinok IS
    FORALL o:Obj
    ( netinfull & o = netin =>
        EXISTS g:Guard EXISTS l:Lev
        ( g << THIS &
          g.fromnet(o,l) &
          l <= max )
    );

  INVARIANT filter USING netinok IS
    FORALL o:Obj
    ( touser(o) =>
        EXISTS g:Guard EXISTS l:Lev
        ( g << THIS &
          g.fromnet(o,l) &
          l <= max )
    );

  INVARIANT netoutok IS
    FORALL d:Subj FORALL o:Obj FORALL l:Lev
    ( netoutfull & d = dest & o = netout & l = max =>
        l = max &
        EXISTS g:Guard
        ( g << THIS &
          g.fromuser(d,o) )
    );

  INVARIANT transport USING netoutok IS
    FORALL d:Subj FORALL o:Obj FORALL l:Lev
    ( tonet(d,o,l) =>
        l = max &
        EXISTS g:Guard
        ( g << THIS &
          g.fromuser(d,o) )
    );

  CONSTRAINT OnlyToUser IS
    netinfull & NOT netinfull' =>
      touser(netin);

  CONSTRAINT OnlyToNet IS
    netoutfull & NOT netoutfull' =>
      tonet(netout,max);

END Guard;

CLASS LANSystem REFINES System IS
  CONST Clearance(s:Subj):Lev;
  VAR   user(s:Subj):WorkStation(s);
  VAR   guard(s:Subj):Guard(Clearance(s));

  MAP Classification(o) INTO Clearance(Originator(o));
  MAP ReadAcc(s,o) INTO user(s).knows(o);
  MAP WriteAcc(s,o) INTO Originator(o) = s;

  FORALL s,d:Subj FORALL o:Obj
    SYNC user(s).send(d,o),
         guard(s).fromuser(d,o);

  FORALL d:Subj FORALL o:Obj
    SYNC user(d).receive(o),
         guard(d).touser(o);

  FORALL d:Subj FORALL o:Obj FORALL l:Lev
    SYNC guard(d).fromnet(o,l),
         EXISTS s:Subj guard(s).tonet(d,o,l);

  INVARIANT Origination
       USING WorkStation.knowledge,
             Guard.filter,
             Guard.transport
```

```
    IS FORALL s:Subj FORALL o:Obj
       ( user(s).knows(o) =>
           EXISTS sys:LANSystem
           ( sys << THIS &
             sys.user(Originator(o)).write(o)
           )
       );

    INVARIANT SimpleSecurityCondition
          USING WorkStation.knowledge,
                Guard.filter,
                Guard.transport
    IS FORALL s:Subj FORALL o:Obj
       ( user(s).knows(o) =>
           Clearance(Originator(o)) <= Clearance(s) );

  END LANSystem;

END lan;
```

ISSUES ON THE DEVELOPMENT OF SECURITY RELATED FUNCTIONAL TESTS

Cornelius J. Haley
Frank L. Mayer

DoD Computer Security Center
9800 Savage Road
Fort George G. Meade, Maryland 20755-6000
(301) 859-6044

ABSTRACT

The Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) requires that a vendor create and use security relevant functional tests. A functional test examines the behavior of a system's user visible interfaces. That is, when a system is tested through its interfaces, the interfaces are expected to perform those functions, and only those functions, for which they are designed. This paper will discuss issues which must be addressed when developing a functional test plan to meet the TCSEC functional testing requirement.

DISCLAIMER

The views contained in this paper are exclusively those of the authors based on experience gained as operating system security evaluators at the Department of Defense Computer Security Center. This paper does not necessarily represent official policy of the Computer Security Center.

INTRODUCTION

The Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) requires that vendors create and use security related functional tests as part of their overall development process and maintain adequate documentation of the test plan. A security related functional test plan should examine the behavior of a system's Trusted Computing Base (TCB) through its user visible interfaces. (TCB is defined by the TCSEC as "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." [sic]). That is, when a TCB is tested through its interfaces, the interfaces are expected to perform those functions, and only those functions, for which they are designed. This use of testing provides a system developer some assurance that the system performs as designed. In the scope of the TCSEC, the use of security related testing provides the DoD Computer Security Center (the "Center") a level of assurance that the TCB properly implements the security policy reflected in the design documentation.

Historically, testing has been an area of concern for system developers. Such questions as 'How much testing and documentation is sufficient?', 'What parts of the system need testing?', and even 'What are functional tests?' are common. This paper discusses issues pertaining to the development and maintenance of a security related functional test plan sufficient to meet the Center's functional testing requirement. However, it is important to note that though this paper will deal specifically with security related testing as defined in the TCSEC, these issues may also be relevant when testing the non-security related portion of a system.

In any discussion in the field of computer science, the lack of common terminology has traditionally been a problem. Therefore, the terms **'feature'** and **'security properties'** need to be introduced at this time. A feature, as used in this paper, is the subject of a test. For example, a feature could be a system's discretionary access control mechanism, a module of code or simply a TCB interface to read the attributes of a file. In other words, a feature is what a test will examine and the scope of a feature depends wholly on the scope of the test itself.

Security properties are defined as the characteristics of a system that are relevant to the TCSEC. These properties include object reuse, labeling, discretionary access control (DAC), mandatory access control (MAC), identification and authentication, auditing, and trusted path. Explanations of the policies and requirements behind each of these security properties are found in the TCSEC.

SECURITY RELEVANCE

Security related functional tests examine all features of a TCB with respect to the security properties of the system. For instance, when examining the TCB with respect to the security property of auditing, all features which generate audit messages, collect audit messages, maintain audit logs, or retrieve information from the audit logs are security relevant and must be tested. The TCB must be examined with respect to all other security properties of the system to determine the features which must be tested.

All aspects of a TCB feature being tested that affect (or are affected by) a security property must be tested. The aspects of a TCB feature that are determined not to be relevant to any security property of the system need not be tested to meet the TCSEC functional testing requirement. However, adequate documentation as to why any portion of a TCB feature is not security relevant should be maintained.

For example, when looking at the discretionary access control (DAC) properties of the TCB interface to change the name of a file, the check for proper access to the file is clearly security relevant. However, the check that the new name satisfies a syntax requirement may not be relevant to the DAC security property and would not need testing.

Ideally, the system designers will determine which features are security relevant during the design phase of a system and document their findings in the system design documentation. However, experience has shown that security testing is not always foremost in a designer's mind. Therefore, the tester may have to examine user documentation and even source code to determine the security relevant features of the TCB.

COVERAGE

One of the initial decisions that must be made when designing functional tests is how extensively to examine a feature (i.e., How much testing is enough?). Some of the more popular methods of testing described below will each yield entirely different levels of coverage. One method of testing is the so-called "exhaustive testing", or the testing of all possible inputs to a feature. This method, while providing a great deal of assurance of the correct operation of the feature being examined, is impractical for use in testing operating systems due to the vast amount of resources required. For example, when examining a system with eight hierarchical access levels and twenty-nine non-hierarchical categories (as the TCSEC recommends), there will be 2**32 (over 4 billion) possible security levels. Testing all 4 billion possible security levels in relation to every relevant feature of the TCB is impractical and uneconomical.

Another method of testing requires that all possible paths through the code be tested. In this method, one need not test all possible inputs to a feature, but simply a subset of the possible inputs that will cause all code paths to be executed. This method allows one to make absolute statements about the security properties of a feature being tested. However, for all but very small or simplistic pieces of code, this method again is impractical due to the complexity of determining the inputs needed to cover all possible code paths and the time required to do the testing.

A more practical method of testing an operating system requires that only a small set of all possible inputs be tested. However, this set of inputs must be representative of all possible inputs. Usually this method will not execute all possible code paths. Rather, it will cover a range of inputs and stress the critical boundaries of a feature. To ensure an adequate set of representative inputs, one should determine a critical boundary of a feature and choose test data on both sides of this boundary. (Determination of critical boundaries will be discussed later in the Test Cases section.) For example, when testing the DAC properties of the TCB interface to read the attributes of a file (e.g., maximum length, last time used, size) a critical boundary would be the fact that the user reading the attributes of the file must have read access to the file. In order to stress this critical boundary test cases should include: the user has read access, the user has some access but does not have read access, the user has no access, and so forth. In addition, this method would have to examine any suspected critical points of a feature. An example of this is to test areas where errors have been known to exist or can be reasonably expected to exist.

Because the set of tests generated by the latter method is much smaller than either of the other two methods, it can be realistically applied to a larger and more complex set of features (e.g., an operating system). Any of these methods should be sufficient to meet the TCSEC functional testing requirement. However, since operating systems tend to be large and complex code structures, the last method is probably most reasonable for most systems.

TEST REPEATABILITY

There are many reasons why tests should be repeatable. Primarily, the developer will want to reuse as much as possible of the original tests when a system's design is updated or enhanced. Design changes should automatically cause updates to the test plan. However, tests that examine features of the TCB which are not affected by design changes should not be modified. Another reason why repeatable tests are important is due the fact that testing will often discover implementation flaws ("bugs"). Since implementation flaws are errors in the source code and are not flaws in the design of the TCB, the test plan should not require modifications before being rerun after "bug fixes". Further, when the system is presented to the Center for evaluation, the evaluation team will want to rerun selected portions of the vendor's functional tests to verify fulfillment of the TCSEC functional testing requirement.

With these issues in mind, the test designer should design all tests to be repeatable in order to allow tests to run without modification. To this end, test documentation is extremely important. All

tests should contain sufficient information for a third party to rerun the tests with minimum difficulty.

In addition, test developers should ensure tests are not dependent on the transient properties of a feature. Transient properties are those aspects of a system that are unpredictable or arbitrary (e.g., time and date, response time). Reliance on this type of data for determining the success or failure of a test will probably require modifications of the test suite before it can be rerun. If a test can not avoid the use of transient properties, this fact and any special instructions for rerunning the test should be well documented.

TEST CASES

Tests may be thought of as a sequence of actions. Often these actions can be grouped together in nearly independent test cases. Test cases consist of actions which examine the rules a feature is designed to implement. For example, a test case could examine the rule of the TCB interface to inspect the attributes of a file which states that the user reading the attributes of a file must have read access to the file.

Some of the rules governing the designed operation of a feature may be security relevant while others are not. A rule which states that a user must have at least read access to a file in order to inspect the attributes of a file is security relevant. However, a rule which states that the length attribute of a file be converted from octal to decimal before being displayed is not security relevant. All of the rules of a TCB feature which are security relevant must be tested to meet the TCSEC functional testing requirement.

The security relevant rules governing the designed operation of a TCB feature can be considered critical boundaries. As discussed in the Coverage section of this paper, test cases should then be chosen to stress these critical boundary. One rule can produce many test cases. The rule stated above for inspecting the attributes of a file would produce many test cases by varying the user's access permission to a file.

Since test cases are chosen to stress a critical boundary, some test cases will be expected to attempt actions which violate the rules of a feature's design. Test cases which violate a rule will result in the failure of an action. This failure would be the expected result of the test case. In those tests where the expected result is failure, this failure would constitute a successful test case. Tests must be designed to recover from these expected failures. For example, an attempt to inspect the attributes of a file is expected to fail if the user has no access to the file. Since this is an expected failure, it would constitute a successful test case.

For efficiency, tests should also recover from unexpected failures (whenever possible) so that all test cases are executed once testing is started. Unexpected failures can be caused by system updates to correct implementation flaws in the TCB. These updates, while correcting the original error, often cause unanticipated or incorrect effects. These side effects may cause tests examining other features of the TCB to fail. For more productive testing, these unexpected failures should only abort those test cases directly affected and allow the remainder of the test suite to execute.

ENVIRONMENTAL ISSUES

When examining a feature, all tests will require that certain environmental requirements be met. Many of these requirements may be constant throughout a group of tests, or even the entire test suite. For example, all tests must assume such obvious environmental requirements as the appropriate hardware configuration and version of the operating system. However, certain other requirements such as the existence of certain user accounts, a predefined file structure, and so forth may also be constant through out an entire group of tests. These types of requirements may be established prior to running a group of tests and remain constant for each individual test case.

In addition to the constant environmental requirements, most individual test cases will have unique environmental dependencies. As an example, look at the following two test cases which examine the TCB interface that reads the attributes of a file: case one will test that a user may read the attributes of a file if he has read access to the file and case two will test that a user may not read the attributes of a file if he has only write access to the file. Assume that for this group of tests, the user account USER1 and file FILEA have already been created with no access to FILEA for USER1. In these particular examples, the first test case would also require USER1 to have read access to FILEA while the second test case would require USER1 to have only write access. The first test case would have to give USER1 read access to FILEA before it can execute the test. However, when the first test case completes execution, it should restore FILEA to its original condition by removing USER1's access to FILEA. Therefore, the second test case can assume that USER1 has no access to FILEA and would only have to give USER1 write access before executing the test.

Failure to attend to environmental requirements will cause problems during initial testing and may create difficulties in repeating tests. For example, if the first test case in the previous paragraph did not reset the original state of FILEA, the second test case would fail. All environmental

requirements and dependencies should also be well documented in the test design documentation.

DOCUMENTATION

Documentation of a test should specify and briefly describe the TCB feature being examined and state the expected results of each test case. References to descriptions of features in user or design documentation may be acceptable. The test documentation should also contain an overview of the test methodology being used to test the features of the system.

Test documentation should list the security properties which are and are not pertinent for each particular feature. For example, the TCB interface to read the attributes of a file would need to be tested with respect to discretionary access control but probably would not need to be tested with respect to identification and authentication. Both of these facts must be documented.

A list of all assumptions being made about the testing environment should also be included in the test documentation. Examples of these assumptions are: the existence of support programs, the location of inputs, or the existence of a predefined hierarchy of files. If tests must execute in a particular order, that ordering must also be specified in the test documentation.

Documentation should exist describing any test support mechanism used to examine a feature (e.g., tools written solely for testing). An example of such a support mechanism is an interface program used to make subroutine calls from command level. This documentation should be sufficient for someone other than the original designer to understand, enhance, and operate the support mechanisms.

CONFIGURATION MANAGEMENT

Developing and maintaining both the system and the test plan using good configuration management techniques will greatly simplify meeting the TCSEC functional testing requirement. Ideally, the system designer should be responsible for designing both the system and the functional test plan. During the design stage of a system, the coverage and security relevance issues discussed earlier should be addressed and the results documented in the system/test design documentation. When the system design goes to the system programmers for implementation, the test design should go to an independent testing organization to implement the test plan. This allows a check and balance system and ensures short cuts are not taken in the testing process. This same procedure should be used when updates and enhancements to the system occur.

The configuration management system should not only encompass changes to the system, but also changes to the test plan and any automated tool developed to help manage the development and execution of a test plan. In short, proper configuration management will make both the tester and the system security evaluator's job easier and more efficient, and will result in a higher level of trust in the correctness of the TCB.

SUMMARY

This paper discussed several issues important when developing a functional test plan to meet the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) functional testing requirement. Initially, the problem of what is security relevant was discussed to provide guidance on determining which parts of a trusted computing base (TCB) are required to be tested by the TCSEC. The paper then described three different methods of testing, explaining the various levels of coverage provided by each and a view on which method to use when testing a system. Other issues presented include designing tests for maximum reuseablity, structuring tests into independent test cases, awareness of environmental dependencies, and maintaining adequate documentation. In addition, the merit of a configuration management process applied to both the system and the test plan was briefly discussed.

Consideration of these issues when developing a functional test plan will help a developer avoid many problem areas. This paper dwelled exclusively on security related functional testing as required by the TCSEC. However, many of these issues are applicable to system testing in general. This paper did not attempt to give an all inclusive process for developing a functional test plan. Rather, it presented "food-for-thought" for a potential test developer desiring guidance on meeting the TCSEC functional testing requirement.

REFRENCES

DoD Computer Security Center, Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-85, Fort George G. Meade, Maryland, August 15, 1983.

PAPER OUTPUT LABELING IN A DEDICATED SYSTEM RUNNING UNDER MVS

Helmut Kurth
Industrieanlagen Betriebsgesellschaft mbH
Einsteinstr. 20
D-8012 Ottobrunn
Federal Republic of Germany

If classified data are handled in an EDP-system, there exists the problem, that one must label all output, especially printer output with the proper security level. There are three conditions, such a label should meet:

1. It must reflect the correct security level of the printed data

2. It must be tamper proof

3. There shall be no way to circumvent the labeling process

But since one can not completely control the information flow in todays EDP systems, there is in most cases no way to fulfill condition 1). Therefore we weaken this condition to:

1*. The label must reflect a security level, which is equal to or higher than the security level of the printed data.

Our task was, to design and implement a software package which fulfills these conditions for printer output on an IBM /370 compatible computer with the MVS operating system and RACF as access control system. Since one can not associate a security label to objects like data sets or main storage with RACF (the design of MVS makes it impossible to implement a security model like the Bell and La Padula model in a secure way), our EDP system works only at one security level at a time. It's a system working in a dedicated mode. To change the security level, one has to shutdown the system, remove all data on permanent storage and clear main storage before one can IPL the system again (of course using another IPL source). So we perform a procedure known as color changing. With this organisation 1* is automatically garanteed, if all output is labeled with the current security level. This security level is entered at IPL time by the system operator and can not be changed.

When we specified our requirements for design and implementation of a program which fulfills condition 1 (or 1*) to 3 we took the Fundamental Computer Security Requirements specified in the DoD's Trusted Computer System Evaluation Criteria known as the Orange Book as a guideline.

These requirements are:

Requirement 1 - Security Policy

Since we have only one security level at a time, there is no need for a security policy, which is able to control the flow of information between different security levels. But all exported data must be labelled with the current security level in a secure way. By organisational measures we assure, that only persons cleared to the security level of the system can access the system. An access control system (in our case RACF) controls and monitors the access to data sets. This system also protects critical system data sets and the data sets used by our paper output control program against unauthorized access.

Requirement 2 - Marking

As we mentioned above, only those objects, which are exported from the system have to be marked. This is especially true for paper output. All paper output must have a label, which reliably identifies the security level of the data on it. Therefore every page of paper output will get a special label at it's top. This label contains the security level, the name of the job that produced the output, the name of the user that started the job, date and time of printing and two numbers. The first number is the current page number relative to the start of the job's output. The second number is a control number, which serves as an additional security measure to detect attempted deceptions or faults in either hardware or software. Before the paper is used in the system, every page is already uniquely labeled by a consecutive number. Every time the system is brought up and every time the paper is changed, our program will ask the operator to type in the number of the first page in the printer. The program will increase this number for every new page and print it as a control number into the label of the page. So, if the control number and the preprinted number don't match a fault or an attempted fraud has occured.

Requirement 3 - Identification

We must properly identify the originator of the output to produce a correct label and to perform correct accounting. This identification is done by the operating system and the access control system. Identification information such as the jobname or the username is automatically passed to our program by the operating system through the used interfaces. Our program copies these information to the output label and to the accounting record.

Requirement 4 - Accountability

Our program builds for every printer output an accounting record, which contains the following information:
the security level of the output, the number of pages printed, the control number of the first page, the control number of the last page, the jobname, the username, date and time the output was printed. So each page of output can uniquely be identified.

Requirement 5 - Assurance

The program is divided into independent sections. Each section operates as a task under MVS. Together with the external writer all these tasks form a single address space. So it is possible to test every program section separately. With this structure all tasks can monitor the execution of the other tasks and check, if they work properly. The control number is another feature to detect a fault or tamper. Since we use only documented interfaces to the operating system, the program flow can be easily examined and no major changes are necessary when a new release of the operating system is installed.

Requirement 6 - Continuous Protection

Since our program operates in it's own address space, it is impossible for another program, which operates in the user state, to observe or change the program flow, or to modify the program or it's data. The program can not be started by a normal user, since it resides in an authorized library which is protected against every access by a normal user. It is not possible to address the printer directly and produce unlabelled output. The data sets containing our program and all data sets used by the program are protected by the access control system. Special SMF-records are generated at various points of the program, to help an auditor to keep track of the program's operation.

SPECIFIC DESIGN

A main design aim was, to make as few modifications as possible to the operating system and to use only documented interfaces. So problems with new releases of the operating systems are reduced to a minimum. We achieved this in the following way:

The standard way to produce paper output under MVS is, to alloacte a so called "System Output (SYSOUT) Data Set" to a printer output class. Then you copy the information you want to print to this data set. When you free the data set, a special system program, the "Job Entry Subsystem (JES)" queues this data set to the output queue specified by the SYSOUT output class. Data sets in this queues can be dequeued either by an output writer program internal to JES or by the so called "External Writer". The external writer provides an interface, which may be used to print output in a non-standard way. The installation may provide it's own output writer and job separator programs to do that. When the external writer dequeues a SYSOUT data set, he first calls the installation provided pre-job separator program, which may produce one or more pre-job separator pages. Then the output writer specified at SYSOUT data set allocation time is called, which reads the SYSOUT data set and copies the information to the printer. If no special output writer name was specified, the system supplied standard writer is invoked. To ensure, that our output writer is always called, we replaced the system provided output writer by our own program. An optional post-job separator program is called when the output writer has finished. Since we deactivate the JES output writer at JES installation time, all printer output is spooled to the external writer. The external writer is not started by the operator directly, but it is envoked by a special driver program. This driver program is started by the operator with the Start command. The external writer itself calls our output writer and job separator programs. The output writer is the program, which copies the output data sets to the printer, produces the page labels and counts the printed pages. The driver program provides a special communication area, which is accessible by all parts of our program. It is used to pass information from one program part to another and to synchronize the different tasks. Accounting is done by a special task, which is started by the driver program. This task runs parallel to the external writer and saves the accounting information for each output printed in one of the special account data sets. There are up to nine of those data sets and they are automatically managed by this task.

The information gathered in the account record for a job's paper output are printed on the post-job separator page at the end of the job's output. On the one hand, the user can check these information, on the other hand he can get a receipt for those pages he brought back to the security registration office on a special form, which is printed below the account information. All account records are passed by tape to a minicomputer at the security registration office. A special program on this computer helps the clerks at that office to keep track of all the labeled output.

PROGRAM PARTS

The Driver Program

The driver program is the program started by the operator. It asks for the current security level and the control number of the first page in the printer. The driver then starts the account task and waits until it gets a ready signal from this task. Then the external writer is started. When the external writer is active, the driver program watches the printer to detect any case, where it enters the "not ready state". This is done in the following way:
If the external writer starts printing of a job's output, a busy flag is set in the communication area by our pre-job separator program. When this flag is on, the driver checks the "device not ready" flag in the UCB (Unit Control Block) of the printer every three seconds. When the external writer waits for output to be printed, a "not ready state" is not reflected in the UCB. So our driver program issues a Sense Channel Command to the printer every three seconds and checks the returned sense information to detect a not ready condition. If a "not ready state" is detected, a flag in the communication area is set to indicate, that a paper change may have occured. A subroutine is called which asks the operator, if the control number has changed and writes a record to the SMF-data set. So any manipulation of the printer can be detected. When the external writer terminates, the driver program sends a termination signal to the account task. This task closes the current account data set and terminates. Then our driver program calls a program which saves the account data sets on tape.

The Account Task

The purpose of the account task is, to manage the account data sets and to write the account records to them. The account data sets are DA data sets with a fixed blocklength and fixed size. The first record of each data set contains information about the status of the data set, i. e. how many records will fit into it, the number of records already used and the date and time the data set was last used. Since physical blocklenght is equal to the length of one record, the system performs no internal buffering. So even when a system crash occurs, no account record will be lost and no special recovery mechanism is needed when the system is brought up again. Before an account record is written to an account data set, the program checks, if the pointer to the first unused record in this data set points to a record marked as empty. If this is not the case, the account task will not use this data set. A message is issued to the system console to consult the system programmer and a special SMF-record is generated.

When started, the account task looks for a free account data set and opens it. Then it sends a signal to the driver program to indicate that it is ready for work and enters the wait state. This wait state is terminated by a signal of our pre-job separator program. The account task then copies information like jobname, username, number of first page of output, date and time from the communication area to the account record area. A ready signal is sent to the pre-job separator program and the driver enters wait state again. This wait state is terminated by the post-job separator program. The account task copies the control number of the last page printed to the account record area writes the account record to disk and updates the control information in the first record of the account data set. If this data set gets full, the account task closes this data set, releases the control for it and looks for another free account data set. If it is not able to find a free account data set, a program which copies all account data set to tape is activated. When the program is ready to accept new account information, it sends a ready signal to the post-job separator program and waits for the next job. When the account task gets a stop signal from the driver program, it closes the current account data set and termintes.

The Pre-Job Separator Program

This program is called by the external writer before a job's output is printed. It generates two separator pages and copies information like jobname, username, output class, date and time to the communication area. The program then sends a signal to

the account task and waits for the response signal.

The Output Writer Program

This program is called by the external writer to produce the output. It transforms the input data set record format, record length and control characters to those suitable for the printer and produces the page label for each page of output. The information needed to build this label is passed through the communication area to our output writer program.

The Post-Job Separator Program

This program is called by the external writer after a job's output is cpmpleted. It produces the post-job separtor page and signals the completion of the output to the account task.

The Tape Copy Program

The tape copy program copies all account data sets which are marked as used to a tape. Only standard label tapes can be used. This tapes are protected by RACF. The names of the tapes which may be used for this program must be specified at installation time by entering their names in a special table. When the tape copy program is invoked, it asks the operator to enter the name of the tape he wants to use. The program checks, if this name has an entry in the table, allocates the tape and copies the account data sets to it. Since only standard label tapes are used, the operating system checks, if the correct tape was mounted. All tapes are protected by RACF against access by unauthorized users. The first record of each tape contains a flag, which is used to prohibit overwriting of a tape, which has not yet been read by the minicomputer in the security registration office. When all account data sets are copied to the tape, a SMF-record is generated which contains the following information: date, time, name of the tape, number of records written to the tape. Then all account data sets and all records are marked as empty and can be reused by the account task.

The Initialization Program

This program is used only once at installation time to create and initialize the account data sets.

INSTALLATION AND ERROR RECOVERY

Installation Dependent Parameter

Some parameters may be changed at installation time to meet special installation dependent requirements. These paramters are collected in a member of the macro library used to assemble the programs. This member is copied into every program. The parameters which may be changed are:

The number of account data sets (minimum 2, maximum 9)

The names of the account data sets

The size of the account data sets

Allowed security levels

The name of the tapes which may be used for saving the account data.

The SMF-record number

Actions Taken, when a Paper Change is Needed

If the last page of the paper stack is printed, a paper jam happens or stop button of the printer is pressed, special actions must be taken. In any of these cases, the printer enters the "not ready state". This causes the next write channel command issued to the printer by the basic I/O system of MVS to fail. MVS then issues a sense channel command to get the reason for this failure. From the sense information returned, MVS detects, that the printer is in the not ready state. Then MVS sets the "device not ready flag" in the UCB of the printer and issues an intervention required message to the system console. The output writer itself is in the wait state during all that time and will get no information about the printer stop. So our driver program watches the UCB of the printer and turns on a special flag in the communication area, if the device not ready flag in the UCB turns on. After every I/O operation the output writer tests this flag. If it is on, he stops further printing and calls a subroutine which asks the operator, if the control number has changed (i. e. if a paper change has happened). Then the output writer calls the post-job separator program to generate an intermediate post-job separator page and an account record for the first part of the output. Then a new account record is initialized for the second part of the output. So when the paper is changed, two (or eventually more than two) account records are generated for one output.

Additional Changes to the Operating System

Two additional exit routines ensure, that no unauthorized user is able to allocate the printer and produce unlabeled and unregistered output.

Error Recovery

Every part of our program has it's own error recovery procedures. So for example, an error in the output writer or job separator will not affect the accounting mechanism. On the other hand, any error in the accounting task causes the driver program to terminate the external writer immediately and to save all accounting information on tape. Every error condition detected by any part of our program generates a SMF record containing information which may be useful to identify and locate the error. In some cases a main storage dump is produced also. Due to the construction of our accounting mechanism, even an abnormal termination of the whole system has the effect, that only the accounting information of that output, which is just printed may be lost. So the operator must collect the accounting information for that specific output manually.
To avoid a loss of information due to a disk crash, it is easy to install a double logging mechanism. Only the account task must be changed.

This system is in operation.

What Counts for Success in Computer Security R&D?

Computer security research and development efforts are pursued in many organizational settings and with many diverse goals within the United States. Clearly, criteria for "success" are a function of the given setting in which the R&D takes place and the goals against which the efforts are undertaken. Highly successful Federal Government, industrial, and academic R&D practitioners and managers are represented on this panel. As the biographies below indicate, each brings a wealth of pertinent experience to share. Each panelist's views on a number of key topics are presented. These include:

- How goals, objectives, and priorities are set in each environment.
- Who are the customers? Is the approach varied for different customers?.
- What metrics are used to tell if progress is being made? How do you know when you are done?

The panelists are Mr. Pat Gallagher, the National Security Agency's Senior Representative to the Department of Defense (DoD) and previously Chief of NSA's Secure Communications Systems Development Group; Mr. Steve Lipner, Senior Engineering Manager for Secure Systems at Digital Equipment Corporation; and a representative of the academic community. The panel chairman is Mr. Larry Castro of the DoD Computer Security Center.

PATRICK R. GALLAGHER, JR.

Mr. Gallagher was born in Philadelphia, Pennsylvania, on 16 August 1936. He graduated from Northeast Catholic High School, Philadelphia. He received his Bachelor of Electrical Engineering degree from Villanova University and his Master of Electrical Engineering degree from Catholic University. Mr. Gallagher was a National Institute of Public Affairs fellow at Cornell University in 1967-1968 and has done further graduate work at the University of Maryland and the University of Pennsylvania. Pat and his wife, Eileen, have six sons and presently reside in Laurel, Maryland. Mr. Gallagher was assigned to NSA as an Ensign, USN, in June 1958. He converted to civilian status in June 1961. Mr. Gallagher started his engineering career designing digital machines. He then moved to TEMPEST R&D where he led a group involved in receiver design. After returning from a fellowship year at Cornell, he was responsible for the group that was producing tactical secure voice equipment for Vietnam. This effort was followed by a tour as Chief of the COMSEC Acquisitions Staff where procurement and logistics were of primary concern.

Completion of a study of the COMSEC organization resulted in the creation of the new Office of COMSEC Applications, in which Mr. Gallagher was appointed the Assistant for Systems Engineering. From there, he was appointed Deputy Chief, Office of COMSEC Engineering, to be the principal focus for the final development and preproduction efforts on PARKHILL and VINSON secure voice equipment. In April 1976, he was appointed Chief, Office of COMSEC Applications, where he served to July 1978, at which time he was made Chief of the NSA organization in Japan. Upon his return from Japan in January 1981, he was appointed Chief, Secure Communications Systems Development Group. He was responsible for all COMSEC R&D for the U.S. government. He served in this capacity until appointment to his present position as NSA/CSS Representative - Defense in February 1985.

STEVEN B. LIPNER

Steven B. Lipner is Senior Engineering Manager for Secure Systems in Digital Equipment Corporation's Central Engineering organization. He has been responsible for Digital's research and advanced development activities in computer security since he joined the company in 1981. His department's principal responsibilities are the development of a prototype security kernel (or Class A1 Trusted Computing Base) for Digital's VAX computers and the development of an encryption system. His organization is also responsible for defining a security architecture applicable to all Digital's products, and for providing computer security support and consultation within Digital.

Prior to joining Digital, Mr. Lipner was with the MITRE Corporation where his involvement with computer security began in 1970. While at MITRE he participated in a number of system security penetration tests, managed the development of the first prototype security kernel for the PDP-11/45, and participated in the design of security enhancements for the Multics operating system. Mr. Lipner also contributed to the hardware architecture design for a secure minicomputer and directed the development of the primary mathematical model of computer security in use today.

Mr. Lipner holds Bachelor of Science and Master of Science degrees from the Massachusetts Institute of Technology. He is a member of the Association for Computing Machinery, the IEEE Computer Society, and a member and former Chairman of the IEEE Computer Society's Technical Committee on Security and Privacy.

LAWRENCE CASTRO

Lawrence Castro is Chief of the Office of Research and Development at the DoD Computer Security Center. In addition to directing the Center's R&D efforts, he is chairman of the DoD Computer Security Program Working Group (PWG). This group is charged with producing the RDT&E component of the five-year Defense Computer Security Program. Through this mechanism the PWG coordinates all Defense-sponsored Computer Security research and development.

Mr. Castro has been a member of the professional staff of the National Security Agency since 1965. His assignments have been primarily in the Research and Engineering Directorate, where he has held positions as Project Engineer, Branch Chief, Division Chief, and Office Chief. From 1978 to 1980, Mr. Castro served in a career-development assignment as Assistant Director for Special Intelligence Systems in the office of the ASD(C^3I). Following that assignment, Mr. Castro returned to NSA and served as the Chief of the Office of Plans and Programs of the Tactical Systems Group. In this position he played an active part in NSA's research and engineering activities to improve tactical SIGINT capabilities within the Tactical Cryptologic Program. Mr. Castro was awarded NSA's Meritorious Civilian Service Award for this work.

Mr. Castro earned Bachelor of Science and Master of Science degrees in Electrical Engineering from the Massachusetts Institute of Technology and the degree of Engineer (with concentration in Communications Systems) from the George Washington University. He graduated form the National War College in June 1984. Much earlier in his career, upon receiving his ROTC commission in the U.S. Army, he completed the Infantry Officers Basic Course and the Army Security Agency Basic Officers Course.

ACHIEVING OPTIMAL COMPLIANCE WITH THE DEPARTMENT OF ENERGY SENSITIVE UNCLASSIFIED COMPUTER SECURITY PROGRAM

Larry Martin
Computer Security Program Manager
U.S. Department of Energy
Office of ADP Management
MA-24 Room F-309, GTN
Washington, D.C. 20545

BACKGROUND

On July 27, 1978, the Office of Management and Budget (OMB) issued Transmittal Memorandum No. 1 to Circular A-71. The transmittal memorandum (TM), entitled "Security of Federal Automated Information Systems" required the head of each Federal Agency to respond to OMB with its plan for implementing a computer security program in accordance with the new requirements. In its response, the Department of Energy (DOE) provided an implementation schedule indicating a five year effort to bring all of its sites into compliance.

Computer security was not a new concept to the Department of Energy when OMB issued TM#1. DOE has had an aggressive computer security program to safeguard its classified information for over thirty years. However, the concept of protecting unclassified information was new and, at first, encountered considerable confusion. The obstacle that had to be overcome was a mind set that only classified information was afforded protection and that if information was not classified, it was not worth protecting. When the need was finally understood, what evolved was two separate and distinct computer security programs administered by two different organizations within DOE. The Office of Safeguards and Security retained their responsibility for the classified computer security program. However, the Office of ADP Management, under the Assistant Secretary for Management and Administration, acquired responsibility for the new unclassified computer security program.

The DOE Sensitive Unclassified Computer Security Program is organized into three phases as follows:

- Implementation Phase - develop, implement, and administer a program for safeguarding DOE computer systems and, in particular, DOE sensitive unclassified information.
- Operational Phase - determine and achieve optimal levels of security at each ADP site.
- Maintenance Phase - maintain required optimal levels of security at each ADP site.

IMPLEMENTATION PHASE

In March 1979, the Department of Energy issued a policy directive requiring the development, implementation, and administration of a program for safeguarding DOE computer systems and, in particular, DOE sensitive unclassified information. This directive, DOE Order 1360.2 "Computer Security Program for Unclassified Computer Systems" implemented Office of Management and Budget Circular A-71, TM#1.

In May 1984, the Department of Energy became the first Department in the Federal Government to report successful implementation of Circular A-71, TM#1 to the Office of Management and Budget. Each of DOE's 75 ADP sites comprising major laboratories, power administrations, energy technology centers, and the Strategic Petroleum Reserve, were certified as having appropriate policies and procedures in place and operational. Articles describing DOE's effort appeared in various periodicals.[1,2]

The policies and procedures implemented by the sites address the following elements that comprise the Sensitive Unclassified Computer Security Program:

1. Each site must designate a Computer Protection Program Manager (CPPM) who must be a management official knowledgeable of both computing and security.
2. The sensitivity of each ADP application must be determined and appropriate safeguards must be assured.
3. A Computer Protection Plan must be formulated and updated annually.
4. Protection specifications for new or significantly changed sensitive applications must be approved in writing by the CPPM.
5. A design review must be performed for new or significantly changed sensitive applications with test results certified by the CPPM.
6. Audits and recertification must be performed at least every three years.
7. Acquisition of computer equipment, software, services, facilities, etc. must identify protection requirements.
8. Risk analyses must be conducted:
   - prior to design approval for new installations
   - whenever the facility or hardware/software changes significantly
   - at least every 5 years.
9. All DOE and contractor employees must have appropriate background screening.
10. Data backup and post-disaster recovery procedures must be established, maintained and used.
11. Processing sites providing resource sharing services must comply with the requirements of the DOE Order 1360.2.
12. A computer access log to identify unauthorized access must be developed.
13. Personnel must have a working knowledge of their computer security responsibilities.
14. Computer file contents must be randomly sampled annually.

OPERATIONAL PHASE

During the post-implementation period, the Office of ADP Management provided three guidelines to the DOE ADP sites. These guidelines were issued so the sites could enhance their programs by further addressing the operational aspects of sensitive unclassified computer security.

ADP Internal Control Guideline[3]

In August 1984, the DOE ADP Internal Control Guideline was issued to dispel confusion created by OMB Circular A-123 "Internal Controls" and the Federal Managers Financial Integrity Act. The DOE ADP sites had completed the risk analyses required by OMB Circular A-71, TM#1, during the Implementation Phase and were being asked to conduct vulnerability assessments. Many sites were unable to differentiate between the two requirements. Furthermore, it was not evident where computer security responsibilities stopped and ADP internal control responsibilities started. The guideline described the similarities and differences between the two OMB Circulars and clarified the overlap. In addition, it provided a methodology for conducting an ADP vulnerability assessment. The guideline was widely used throughout the Federal government following its announcement in the Government Computer News.[4]

Security Guidelines for Microcomputers and Word Processors[5]

In March 1985, the DOE Security Guidelines for Microcomputers and Word Processers were issued. There were no references to microcomputers or word processors in DOE Order 1360.2 since it was issued in 1979. The Office of ADP Management recognized the need to issue security guidelines as office automation proliferated throughout the Department. The guideline is directed at the user and assumes little or no prior ADP experience. It is short, to the point, and is easy reading. Its objective is to develop a security mind set at the time a user first becomes familiar with the office automation equipment. This guideline was widely requested following an announcement in the Government Computer News.[6]

Sensitive Unclassified Computer Security Program Compliance Review Guideline[7]

The Department developed and issued this guideline for ADP site management and program monitors to evaluate the effectiveness of a site's program. The DOE Sensitive Unclassified Computer Security Program Compliance Review Guideline enables the user to determine the level of computer security required by the site based on vulnerability and sensitivity and to ascertain whether the site has achieved the required level. If the site has not achieved optimal security, the guideline provides the basis for sound recommendations which when implemented will improve a site's security to the required level.

DOE Order 1360.2 requires the Operations Offices to conduct annual on-site compliance reviews of DOE and DOE contractors to assess the effectiveness of the computer security program procedures used and to document the results of the reviews. The guideline provides effective tools to perform both of these required functions. It is not acceptable to simply check to see if a site has policies and procedures, it is necessary to evaluate the effectiveness of those procedures and recommend improvements, as appropriate.

The objectives of the guideline are:

- To provide a means to determine the required level of security for DOE ADP sites based on vulnerability and sensitivity;
- To provide descriptive targets which define the elements of optimal computer security achievable in practice with present technology;
- To provide an objective format for evaluating the security procedures currently implemented in response to DOE Order 1360.2.

These objectives and executed in a three step methodoloy during the compliance review, as follows:

Step 1: Establish ADP Installation Security Levels. The extent to which an ADP installation achieves optimal ADP security depends upon the vulnerability and sensitivity of the installation and applications. Many installations need only achieve minimum security procedures, while other installations must develop and enforce more stringent security. The reasons for these differences in approach include not only application sensitivity, but also perceived risk and value by the public, the media, or by unauthorized outsiders. Recognizing the need for diversity, DOE has established the following 3 ADP installation security levels which we have designated as indicated:

- Low Security (Type A installation)
- Moderate Security (Type B installation)
- High Security (Type C installation)

The section of the guideline that identifies the ADP installation security levels provides an objective method for determining a site's appropriate security level. The assessment of a site's security level is accomplished by answering a limited number of questions relating to vulnerability. An analysis of the answers indicates the appropriate security level. However, flexibility is built in with the human factor. In the event that the analysis of the assessment questionnaire indicates a security level that common sense would otherwise indicate as either too high or too low, the level can be adjusted accordingly by mutual agreement between site management and the Operations Office Computer Protection Program Coordinator.

Step 2: Identify Goals for Optimal Security. Once an appropriate security level has been assigned to an ADP installation, objectives are established to achieve optimal security. There are twelve target areas for which specific security objectives and goals are established. These are:

- Applications Development and Design
- Data Handling
- Disaster Recover Plan
- Hardware
- Input/Output
- Internal Audit

- o Microcomputers
- o Personnel
- o Physical
- o Policies and Procedures
- o Remote Job Entry
- o System Software

These 12 target areas are cross-referenced to the 14 requirements of DOE Order 1360.2 to assist the reviewer defining the site's goals for achieving optimal security.

Step 3: Complete Questionnaires. Finally, a set of security review questions are provided for evaluating the computer security program at a DOE site. The questionnare responses are intended to be interpreted by an experienced computer system professional from the DOE operations office performing the compliance review. The questionnaire format was designed to provide adequate information to both the security program reviewer and the reviewee. Codes are designed to be simple and useful as practical.

Upon completion of the detailed questionnaires, the reviewer can evaluate the effectiveness of the existing security program. Then based upon the level of security required by the site and the goals established for optimal security, a final determination is made. The site is either declared in compliance with optimal security or it is declared less than optimal. If declared optimal, the site moves into the third and final phase of the program.

Although proven to be effective tools during field tests at two DOE contractor ADP sites, the techniques and questionnaires in the guideline are presented for consideration only. The CPPC at each Operations Office may use this Guideline "as is," to develop his/her own compliance review criteria, or may select a completely different approach. However, in any event, it is mandatory that the effectiveness of the site's policies and procedures be evaluated and, when appropriate, sound recommendations be made for improvement. The DOE goal is for each site to achieve an optimal level of computer security before moving into the Maintenance Phase and the guideline provides a path to that end. In the event that an alternate method review is chosen, the method must be reviewed and approved by the Office of ADP Management before it is adopted.

Traditionally, the unclasssified computer security program has been costly in manpower and budget, requiring that any resources needed be taken out of a site's overhead. The Office of ADP Management recognized the level of burden this program places on the field offices and realized that in some cases, reallocation of resources to other priorities could hamper the best of intentions by the field offices to perform these on-site compliance reviews. The result could easily be a list of excuses about why the reviews were not conducted. However, due to the commitment of DOE management to a strong and effective sensitive unclassified computer security program and the major importance that the Department places on protecting its computers and sensitive information, the Office of ADP Management has arranged contractor support and offered these resources at no expense to the Operations Offices. In other words, operations offices are given the resources to perform the required reviews, if they so choose.

If the site is determined to be below the optimal level, recommendations based on the security goals established during Step 2 of the review are provided to site management in the compliance review report. The Operations Office will track site progress on the implementation of the recommendations against the site's milestone plan. Upon completion, a follow-up review is performed to verify implementation. The site has achieved optimal compliance with the Department of Energy Sensitive Unclassified Security Program.

MAINTENANCE PHASE

The final phase of the program requires the sites to maintain the optimal level of computer security achieved earlier in the program. As technology changes, site management must continue to update its sensitive program to maintain the state-of-the-art. During this phase, the Operations Offices will review each site once every two years to ensure continued compliance. The Office of ADP Management will continue its oversight role of the Operations Offices and provide guidance and assistance, as appropriate.

CONCLUSION

The Department of Energy Sensitive Unclassified Computer Security Program is unique and demonstrates the commitment of DOE management toward the protection of DOE computers and sensitive unclassified information. The program is structured but has built in flexibility to allow for the diversity of sites with a variety of missions. DOE has been aggressive in its attempt to comply with OMB Circular A-71, TM#1. The Department anticipates changes to Federal policy with the issuance of National Security Decision Directive 145[8] and the revision of OMB Circular A-71, TM#1, as Appendix III of the draft OMB Circular entitled "Management of Federal Information Resources."[9] As policies change, DOE will continue to strive for excellence in the protection of its sensitive unclassified information and computers which are vital to the performance of its missions and functions.

ACKNOWLEDGEMENTS

Major contributions to the Department of Energy Sensitive Unclassified Computer Security Program and ADP Internal Control Guidelines were made by Arthur Young and Company and the Los Alamos National Laboratory.

The author is indebted to Kathryn Clay Martin who provided valuable guidance and suggestions during the development of this paper.

REFERENCES

1. "DOE Achieves Computer Security Milestone," Government Computer News, p.21, Vol. 3, No. 6, June 1984.

2. "Department of Energy Achieves Computer Security Milestone," Computer Crime Digest, pp. 4&5, Vol. 2, No. 8, August 1984.

3. DOE/MA-0165, ADP Internal Control Guideline, U.S. Department of Energy, August 1984.

4. "Energy Department Has New Internal Control Plan," Government Computer News, p. 75, Vol. 3, No. 12, November 1984.

5. DOE/MA-0181, Security Guidelines for Microcomputers and Word Processors, U.S. Department of Energy, March 1985.

6. "DOE Issues Security Guidelines," Government Computer News, p. 68, Vol. 4, No. 7, April 26, 1985.

7. DOE/MA-0188, Sensitive Unclassified Computer Security Program Compliance Review Guideline, U.S. Department of Energy, June 1985.

8. National Security Decision Directive 145, "National Policy on Telecommunications and Automated Information Systems Security," The White House, September 17, 1984.

9. Draft OMB Circular A-____, "Management of Federal Information Resources," Office of Management and Budget, Federal Register, pp. 10734-10747, Vol. 50, No. 51, March 15, 1985.

Development of a Multilevel Secure Local Area Network

D.D. Schnackenberg

Mail Stop 8H-35
Boeing Aerospace Company
P.O. Box 3999
Seattle, WA 98124

Boeing Aerospace Company is developing a multilevel secure (MLS) local area network (LAN), designed to meet the A1 criteria of DoD Trusted Computer System Evaluation Criteria (1). The development effort is funded under internal research and development. This paper will present an overview of the MLS LAN development, and will discuss security design issues (e.g., protocol security), security architecture, security policy, formal security policy model, current status and future directions for our MLS LAN.

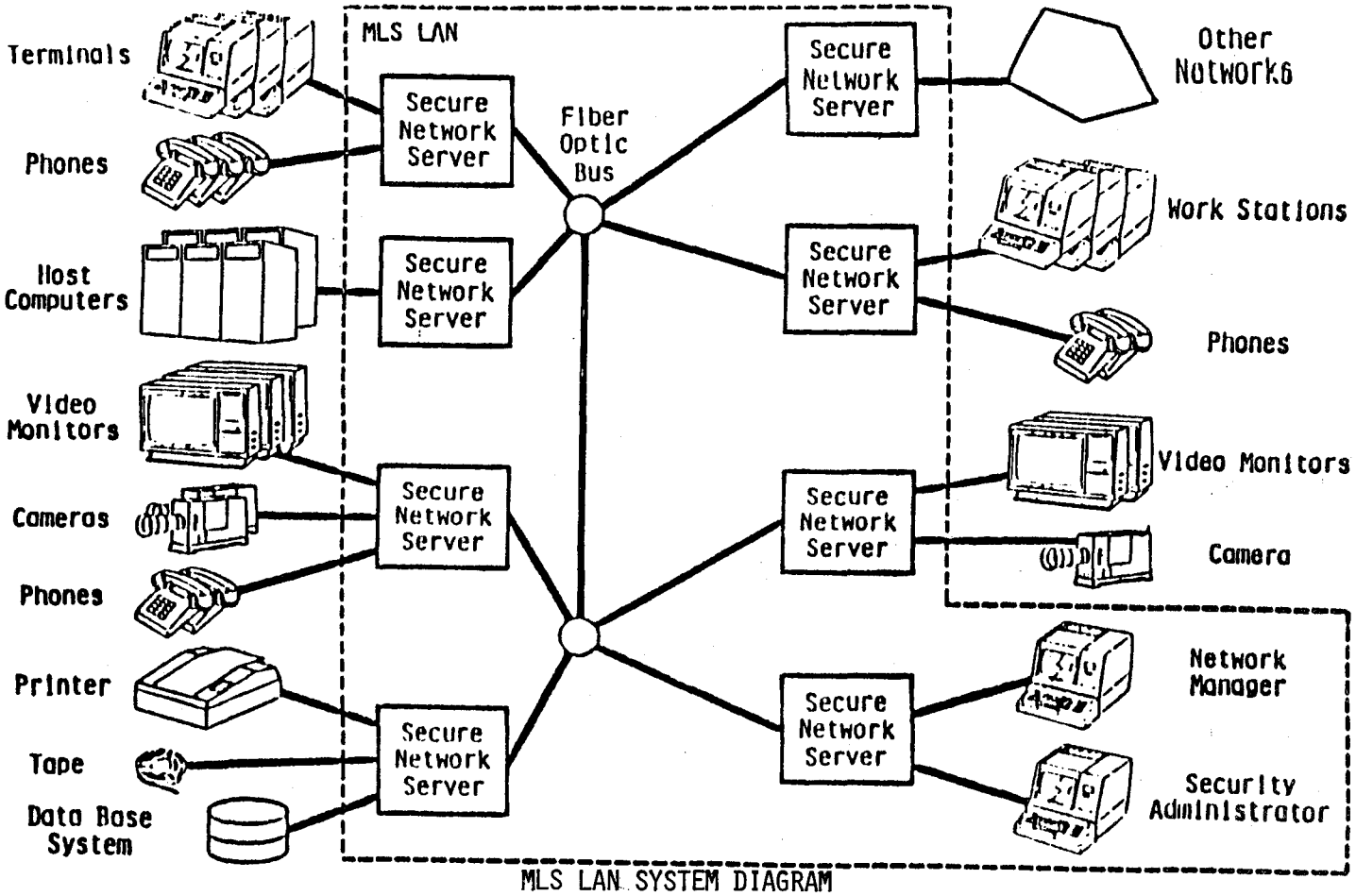
OVERVIEW

The MLS LAN is a high performance network that supports simultaneous transmission of digital (150 Mbps), voice and analog video data, using a wavelength division multiplexed fiber optics communications medium. Figure 1 shows the system diagram for the MLS LAN. The MLS LAN comprises network access units (called Secure Network Servers or SNSs) and a management node. The network trusted computing base (TCB) is distributed across the SNSs and the network management node. The SNS provides network security functions to ensure that data is not sent or received at an inappropriate sensitivity level for the subjects the SNS controls. The network management node provides the administrator interface to the network, and maintains the network configuration and security databases. The SNS provides embedded upper layer protocols (e.g., Transmission Control Protocol (TCP), TELNET and File Transfer Protocol (FTP)), supports the connection of terminals to the network, and will eventually support embedded user services (e.g., file server and mail server).

There are 5 types of users of the network:

1. human users on a network terminal (or workstation configured as a terminal);
2. host processes acting on the behalf of some human user (the network view of host processes is a host-to-network logical channel);
3. voice devices;
4. analog video devices; and
5. high bandwidth digital stream devices.

The demonstration system will provide four optical channels on the fiber optics medium. The terminal, host and voice traffic is transmitted using one of these channels. Access to this channel is gained using the IEEE token passing bus protocol. The remainder of the channels are used to transmit circuit-switched analog video and high bandwidth digital stream data. The circuit switching is controlled by users at terminals through the digital network.



MLS LAN SYSTEM DIAGRAM
Figure 1

An SNS comprises multiple 286s, hardware for the subscriber interfaces, link control hardware and a fiber optics interface. A multi-CPU executive provides dynamic domains for the tasks within an SNS across multiple CPUs. This executive acts as a separation kernel, and enforces a policy of complete isolation of user data streams.

SECURITY DESIGN ISSUES

The security design goals include providing a uniform interface to hosts, excluding upper layer protocols from the TCB, and developing a uniform security policy and model independent of the protocol set and subscriber device (e.g., host, terminal, or video camera).

The philosophy for the MLS LAN is to provide sufficient protection mechanisms supporting separation of host process communication, as specified by the host. The goal is to ensure that communication between host processes meets the mandatory and discretionary policies specified in CSC-STD-001-83 at the system level. To meet this requirement, access control to the network is provided at or above the transport layer. Within the network, complete separation of communication objects (datagrams, connections, and sessions) is provided. This implies that network software outside the TCB be dedicated to a single communication object (connection or session). To support separation of host process connections, the TCB assumes responsibility for all addressing within the network, so that correct delivery is assured by the TCB.

The network makes some security assumptions about the host in areas that are outside the scope of network control. It is the responsibility of the security administrator to ensure that only hosts with these characteristics are attached to the network. The following assumptions are made:

1. each host will authenticate its users, and will provide the network with the correct user identity when a network service is requested on behalf of the user;
2. each host will correctly label packets sent to the network within the range that the host is assigned;
3. the host will control the process-to-process covert channels available through the modulation of network resources;
4. each host provides adequate protection for data within the host's range, and provides adequate discretionary and mandatory access controls for the users of the host;
5. the host will perform access checks based on the network provided user identity when a remote user is attempting to gain access to host resources through the network (i.e., the network does not control access to host resources, but rather to the host); and
6. the host will control host process utilization of the network, so that host processes cannot deny access to other host processes.

The software architecture allows the major part of TCP and all of TELNET to reside outside the TCB. To limit the size of the TCB, each TCP connection is single-level. However, multiple concurrent TCP connections can be active within an SNS at differing security levels. The TCB not only separates data by security level, but also provides complete isolation of user sessions. This separation is enforced at both the network and host-to-network interfaces, and within the SNS. The User Datagram Protocol (UDP) is provided as a multilevel service to hosts. The design supports reliable file transfer (through FTP) between hosts at different security levels. To provide this capability requires that a major part of FTP resides within the SNS. The data transfer between the SNSs is reliable (uses TCP). When the destination host is at a higher sensitivity level than the FTP session, there would be limitations placed on the control information passed from the host to the non-TCB FTP functions.

An issue that arises at the system level is the provision of end-to-end user identification and end-to-end trusted path. The network must be able to support these requirements. This implies that the TCB components must be capable of detecting modification of information passing through an end-to-end trusted path.

NETWORK TCB

Figure 2 shows the network TCB. The TCB includes the SNS security functions, the network management software, the executive, and the network hardware.

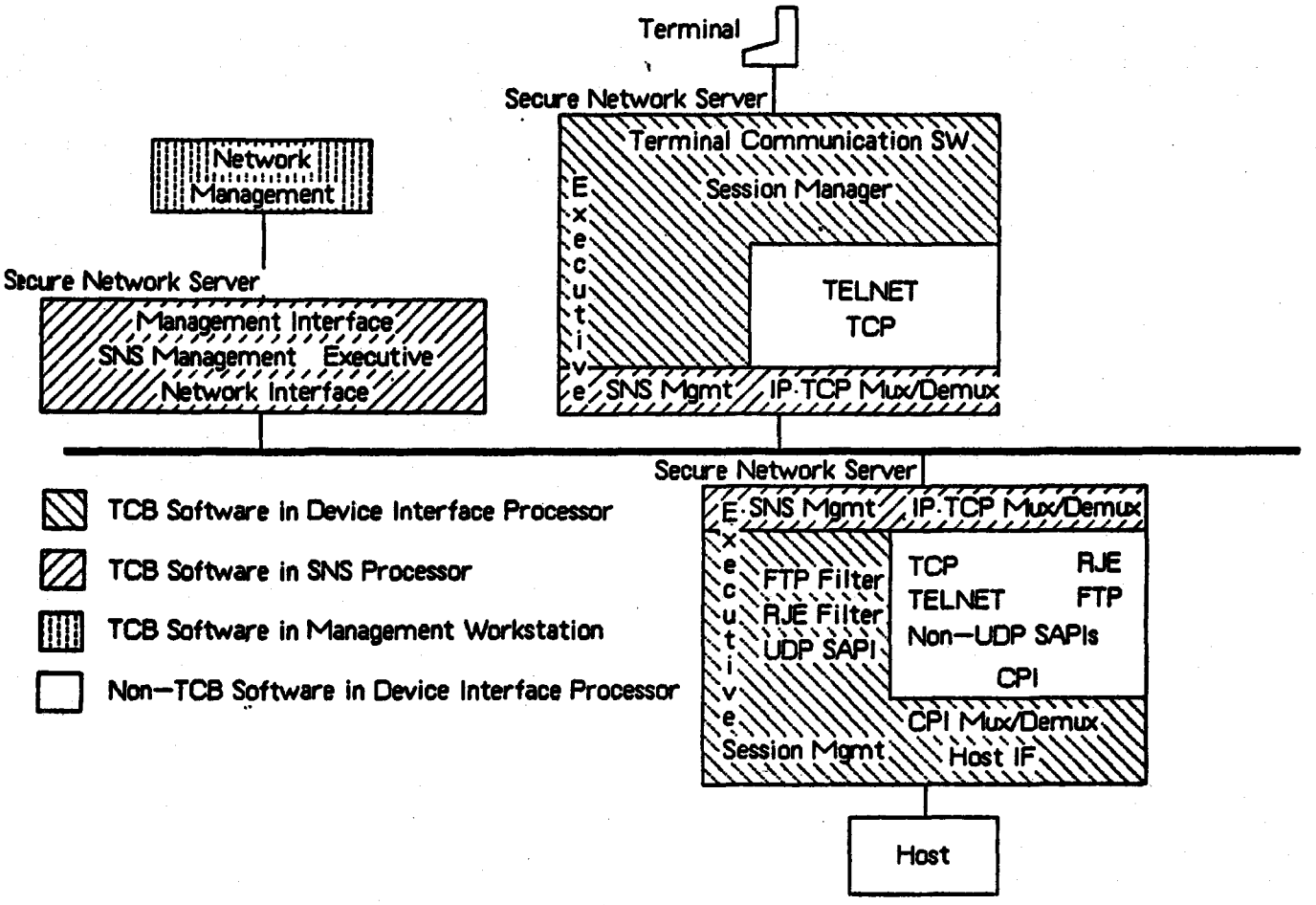
The SNS security functions include the access control functions, and support functions for network management and security control. These include executive, multiplexing/demultiplexing, access control, session management, startup, shutdown, audit and performance monitoring. Network management functions include auditing, performance monitoring, network and security administrator support, network configuration and reconfiguration, authentication, maintenance of access control tables, recognition and notification to the security administrator for security alarm conditions These functions are provided in a dedicated network device that is assumed to be physically protected from unauthorized tampering.

Within an SNS there are two or more Intel 286 microprocessors. The tasks (Intel 286's concept of process) in these processors provide the network protocol and management services. These tasks are controlled by an executive, which provides memory management, task management, timer management, signal management, and intertask communication services to these tasks. For intertask communication and signals, the communicating tasks may or may not be within the same processor of the SNS. This is transparent to the tasks.

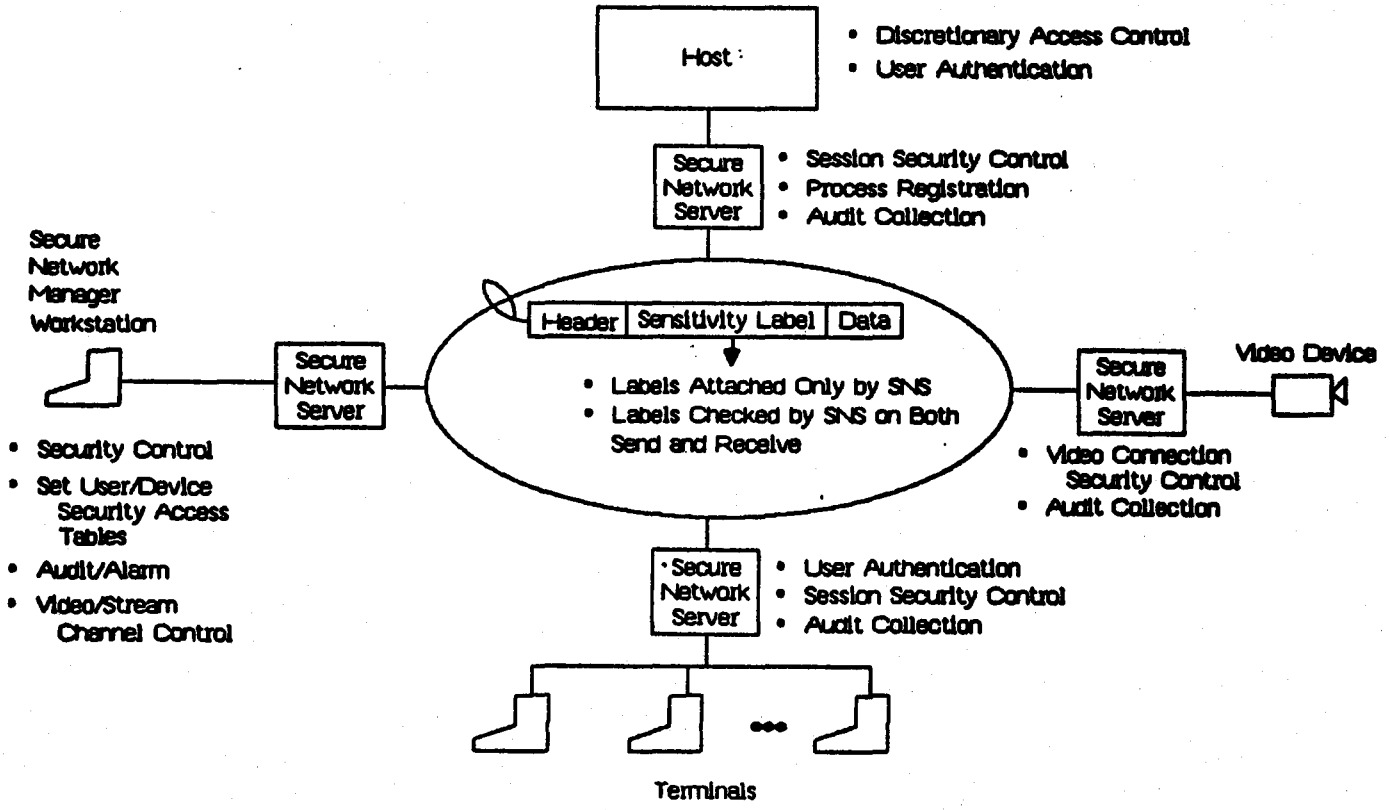
SECURITY DESIGN

The network level security design is illustrated in figure 3. Each packet placed on the network has an attached sensitivity label provided by the TCB. When a packet is received at an SNS, the TCB checks to make sure that the packet destination is active, and is permitted to receive a packet at the provided sensitivity level. For terminals attached directly to the network, the SNS provides login and authentication services. The overall network security control resides at the network management node.

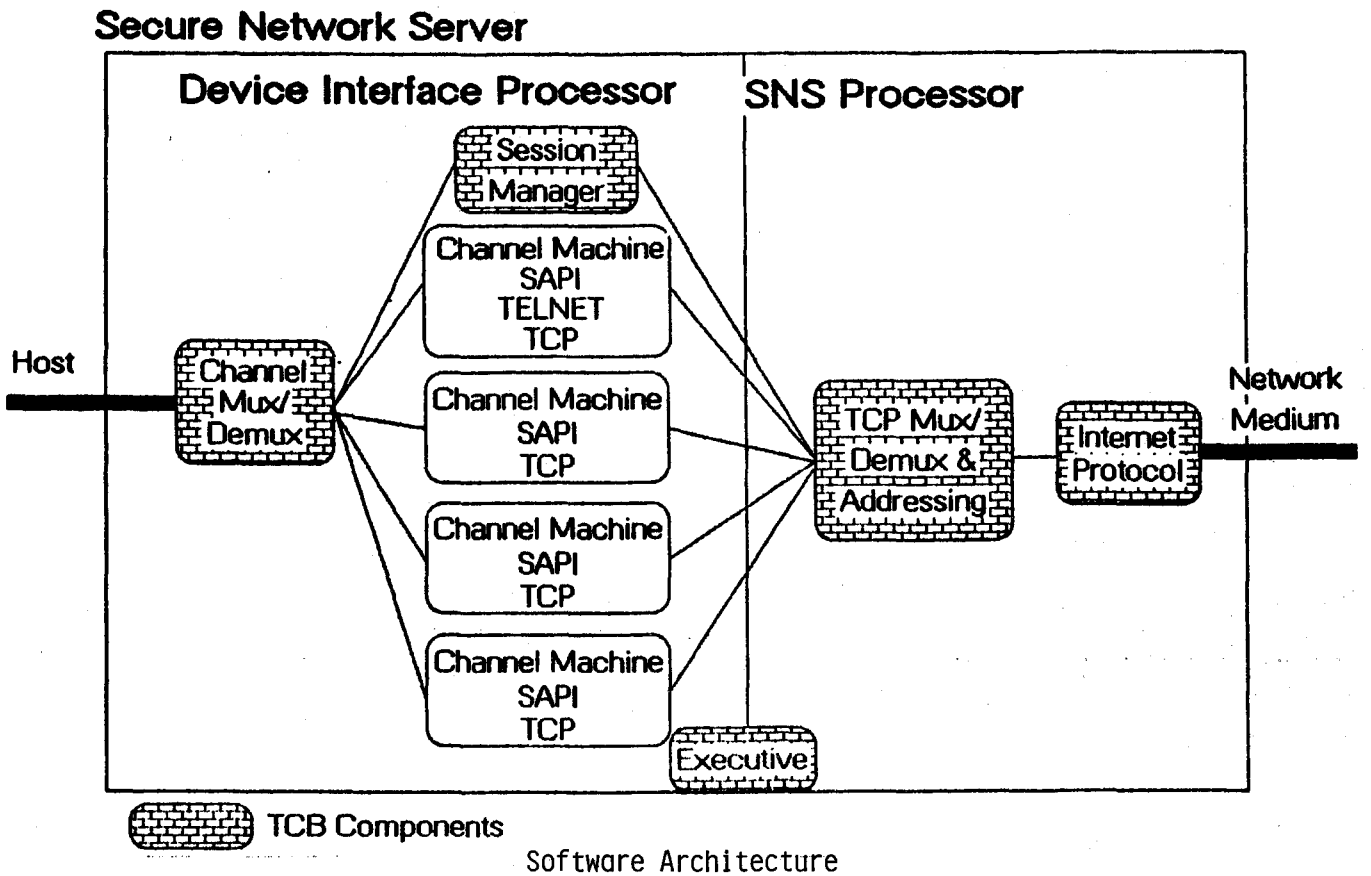
Within an SNS the executive acts as a separation kernel (3). It enforces complete separation of the non-TCB processes that support different user sessions. Figure 4 illustrates the software architecture for the SNS. Each active service (e.g. TCP) is logically separated by the TCB from all other services within an SNS. There is a different processing stream for each active service. This is similar to the software architecture for the Communications Operating System Network Front End (2). The separation of these services is supported by the multiplexing/demultiplexing functions in both TCP and the WWMCCS HFE protocol. The session managers enforce the access control policies at the interface between the network and external devices, and control the creation and deletion of the processing streams that perform the non-TCB protocol functions. The security design uses the Intel 286 processor features to provide task separation and separation of TCB from non-TCB functions within an SNS.



Trusted Computing Base
Figure 2



MLS Design Approach
Figure 3



Software Architecture
Figure 4

SECURITY POLICY

The policy and model for the MLS LAN address the control of access to network communication objects by network subjects. In addition to the external subjects listed above, there are internal network subjects -untrusted tasks performing communication functions. The policy for these internal subjects is complete isolation of processing for different user sessions.

The policy enforced at the external interface to the network is an extension of the policy defined in the DoD Trusted Computer System Evaluation Criteria for Class A1. The extensions address multilevel subscribers and multilevel services for these subscribers.

Discretionary Access Control Policies

The policies enforced by the network TCB at the network external interface are as follows:

1. if a subject has send access to a communication object, then the communication object and subject are active and the subject has send discretionary access rights to the communication object; and

2. if a subject has receive access to a communication object, then the communication object and subject are active and the subject has receive discretionary access rights to the communication object.

Mandatory Access Control Policies

The policies enforced by the network TCB at the network external interface are divided into two cases: multilevel subjects and single-level subjects. In either case, each subject is associated with a device, and the maximum and minimum sensitivity levels for a subject must be within the range of levels for the associated device. The multilevel subjects (currently UDP processes are the only multilevel subjects) are permitted concurrent access to multiple communication objects at different sensitivity levels. The single-level subjects are given a range of sensitivity levels, but can connect to only one communication object at any time. This is used to model services such as a TELNET server on a multilevel host. The server is announced to the network by the host along with the range of sensitivity levels that the host is willing to support for the server. The TELNET service (and server) is single-level, so that when a TELNET session request is received at the SNS, the session sensitivity level must be within the range for the server. The server's sensitivity level will be the same as the session level for the duration of the session.

Multilevel Subjects

1. If a subject has send access to a communication object, then the communication object is active and the communication object's current sensitivity level is within the subject's range of sensitivity levels.

2. If a subject has receive access to a communication object, then the communication object is active and the communication object's current sensitivity level is at or below the subject's maximum sensitivity level.

Single-Level Subjects

1. If a subject is single-level, then the subject's current level is between it's maximum and minimum levels.

2. If a subject has send access to a communication object, then the communication object is active and the communication object's current sensitivity level is equal to the subject's current sensitivity level.

3. If a subject has receive access to a communication object, then the communication object is active and

the communication object's current sensitivity level is at or below the subject's current sensitivity level.

FORMAL SECURITY POLICY MODEL

The principle state components in the MLS LAN security model are the attributes of the active communication objects and subjects. Of particular interest are the sensitivity levels of the objects and subjects, and the discretionary access control lists for the objects.

The system may change state when a subject requests activation of a communication object, a subject requests access to a communication object, a subject requests disconnection from a communication object, a subject requests that subjects be added to a communication object's discretionary access control list, a communication object becomes inactive, a subject becomes active, or a subject becomes inactive. When a subject requests communication object activation, that subject becomes the owner of the communication object. The owner is the sole subject able to modify the discretionary access control list. When the communication object is initialized, the subject specifies a sensitivity level for the object (or this can be set to the subject sensitivity level if the subject is single-level), and an initial list of subjects and modes of access for the discretionary access control list. The specified level must be within the subject's range. The discretionary access control list will typically be specified as the remote socket (or logical name for a remote process) for a TCP connection. The state invariants (Ina Jo criterion) for the model are formalizations of the policy defined above. When a subject attempts to connect to a communication object, the state invariants are enforced to ensure that the subject has the appropriate authorization for that object.

When a subject becomes active, a range of sensitivity levels is set for that subject, and this range must be within the range specified for the associated device. For host processes, the host TCB is responsible for providing the network with this range of sensitivity levels. This range must be within the limits placed on the host by the network security administrator.

Excerpts from our formal security policy model written in the Ina Jo language are included as an appendix to this paper. The part of the model presented describes the policy enforced at the network external interface. The formal verification approach is (1) to use the Ina Jo tool to prove the theorems required for the model, (2) to develop the formal top-level specification (FTLS) required by the Criteria as an Ina Jo second level specification, and (3) to use the Ina Jo tool to prove consistency of the FTLS with the formal model.

CURRENT STATUS AND FUTURE DIRECTIONS

We currently have prototype hardware, and a large part of the network software is complete. Network functionality will be added in increments, with a full scale demonstration system completed by the end of 1985.

The major protocol security issue encountered was determining the degree of support provided within the LAN for system-wide session layer security. There is a need for end-to-end user identification and trusted path to support the system level security requirements. The DoD protocol suite does not have a session layer supporting these requirements. We are addressing these issues as part of this research program, and plan to provide this capability as part of our MLS LAN product. We are also investigating the incorporation of encryption into the MLS LAN.

Enhancements are planned for the MLS LAN to approach the functionality shown in figure 1. In 1986, we plan to develop a file server, mail server and gateway to DDN. An effort is underway to address the packaging issues for the MLS LAN. Provision of higher data rates (up to 300 Mbps) is being investigated.

References

(1) DoD Computer Security Center, "DoD Trusted Computer System Evaluation Criteria," CSC-STD-001-83, 1983.

(2) Grossman, G., "A Practical Executive for Secure Communications," Proceedings of the 1982 Symposium on Security and Privacy, Oakland California, IEEE Computer Society, April 1982.

(3) Rushby, J., "A Trusted Computing Base for Embedded Systems," Proceedings of the DoD/NBS Computer Security Conference, Gaithersburg Maryland, September 1984.

Appendix

Formal Security Policy Model for an MLS LAN

Type

```
    Level,
    Ext_User,
    Subject < Ext_User,
    Device <Ext_User,
    Subjects = Set of Subject,
    Comm_Object,
    mode = (send, receive),
    Security_Mode = (sl, mls)
```

Constant

```
    Lteq(Level, Level) : Boolean,
    Resides_On(Subject) : Device
```

Variables

```
    Active_Comm_Object(Comm_Object) : Boolean,
    Send_DAC_List(Subject, Comm_Object) : Boolean,
    Receive_DAC_List(Subject, Comm_Object) :  Boolean,
    Object_Level(Comm_Object) : Level,
    Connection(mode, Comm_Object, Subject) : Boolean,
    Active_Subject(Subject) : Boolean,
    Subject_Mode(Subject) : Security_Mode,
    Current_Level(Subject) : Level,
    Min_Level(Ext_User) : Level,
    Max_Level(Ext_User) : Level,
    Owner(Subject, Comm_Object) : Boolean
```

Initial

```
  A" c : Comm_Object
  (~Active_Comm_Object(c))
  & A" s : Subject, c : Comm_Object
  (~Send_DAC_List(s, c) & ~Receive_DAC_List(s, c) &~ Owner(s, c))
  & A" s : Subject, c : Comm_Object, m : mode
  (~Connection(m, c, s))
  & A" s : Subject
  (~Active_Subject(s))
```

Criterion

```
  A" m : mode, c : Comm_Object, u : Subject
     ( Connection(m, c, u) - >
        Active_Comm_Object(c) & Active_Subject(u)
        & (m = send - >Send_DAC_List(u, c))
        & (m = receive -> Receive_DAC_List(u, c)))

  & A" m : mode, c : Comm_Object, u : Subject
     ( Connection(m, c, u) & Subject_Mode(u) = mls - >
        (m = send - >
          Lteq(Object_Level(c), Max_Level(u))
           & Lteq(Min_Level(u), Object_Level(c)))
      & (m = receive - >Lteq(Object_Level(c), Max_Level(u))))

  & A" m : mode, c : Comm_Object, u : Subject
     ( Connection(m, c, u) & Subject_Mode(u) = sl - >
        & (m = send -> Current_Level(u) = Object_Level(c))
        & (m = receive - >Lteq(Object_Level(c), Current_Level(u)))
        & A" c1 : Comm_Object, m1 : mode (c1 ~ = c - >~Connection(m1,c1,u)))

  A" u : Subject
     (Lteq(Max_Level(u),Max_Level(Resides_On(u)))
        & Lteq(Min_Level(Resides_On(u)), Min_Level(u))
        & Lteq(Min_Level(u), Max_Level(u))
        & (Active_Subject(u) & Subject_Mode(u) = sl - >
         Lteq(Min_Level(u), Current_Level(u))
         Lteq(Current_Level(u),  Max_Level(u))))
```

```
Transform Set_Up_Comm_Object(u : Subject, c : Comm_Object, l : Level,
                                    S : Subjects, R : Subjects) External
```

```
Effect

    Lteq(Min_Level(u), l) &~ Active_Comm_Object(c) & Active_Subject(u)
    & ( Subject_Mode(u) = sl - >
        (A" c1 : Comm_Object, m : mode
          (~Connection(m, c1, u))))
    & A" c1 : Comm_Object, u1 : Subject
        ((N"Active_Comm_Object(c1) <-> c1 = c |Active_Comm_Object(c1))
        & (N"Send_DAC_List(u1, c1) <->
          u1 <: S & c1 = c |Send_DAC_List(u1, c1))
        & (N"Receive_DAC_List(u1, c1) <->
          u1 <: R & c1 = c | Receive_DAC_List(u1, c1))
        & (N"Object_Level(c1) = (c1 = c => l
                                          < > Object_Level(c1)))
        & (N"Owner(u1, c1) <-> u1 = u & c1 = c | Owner(u1, c1)))
    |NC"(Active_Comm_Object, Send_DAC_List,
                Receive_DAC_List, Object_Level, Owner)

Transform Connect_to_Comm_Object(u : Subject, c : Comm_Object, m : mode)
                                                    External

Effect

    ( Active_Comm_Object(c) & Active_Subject(u)
    & (Subject_Mode(u) = sl - >
        (A" c1 : Comm_Object, m1 : mode
            (Active_Comm_Object(c1) & Connection(m1, c1, u) - >c1 = c)))
    & ((m = send &  Send_DAC_List(u, c)
            & Lteq(Min_Level(u), Object_Level(c))
            & Lteq(Object_Level(c), Max_Level(u)))
       | (m = receive & Receive_DAC_List(u, c)
            & Lteq(Object_Level(c), Max_Level(u))))
    & A" m1 : mode, c1 : Comm_Object, u1 : Subject
        (N"Connection(m1, c1, u1) <->
            m1 = m & c1 = c & u1 = u | Connection(m1, c1, u1))
    & (Subject_Mode(u) = sl - >(m = send & N"Current_Level(u) = Object_Level(c) |
       m=receive&Lteq(Object_Level(c),N"Current_Level(u)))
    & A" u1 : Subject ( u1~ = u -> N"Current_Level(u1) = Current_Level(u1)))
    | NC"(Connection, Current_Level)

Transform Disconnect_from_Comm_Object(u : Subject, c : Comm_Object,
                                                    m : mode) External

Effect

    A" m1 : mode, c1 : Comm_Object, u1 : Subject
        (N"Connection(m1, c1, u1) <->
                    (m1 ~= m| c1~ = c | u1~ = u) & Connection(m1, c1, u1))
    | NC"(Connection)

Transform Add_to_DAC_List(u : Subject, c : Comm_Object, S : Subjects,
                                                    R : Subjects) External


Effect
    Owner(u, c) & Active_Subject(u) & Active_Comm_Object(c)
    & (Subject_Mode(u) = sl = >Lteq(Current_Level(u), Object_Level(c))
                                < > Lteq(Min_Level(u), Object_Level(c)))
    & A" u1 : Subject, c1 : Comm_Object
        ((N"Send_DAC_List(u1, c1)< - >
            u1 <: S & c1 = c |Send_DAC_List(u1, c1))
        & (N"Receive_DAC_List(u1, c1) <->
            u1 <: R & c1 = c | Receive_DAC_List(u1, c1)))
    |NC"(Send_DAC_List, Receive_DAC_List)

Transform Release_Obj(u : Subject, c : Comm_Object) External

Effect

    (Receive_DAC_List(u, c) |Send_DAC_List(u, c)) & Active_Subject(u)
    & (Subject_Mode(u) = sl = >Lteq(Current_Level(u), Object_Level(c))
                                < > Lteq(Min_Level(u), Object_Level(c)))
    & A" u1 : Subject, m : mode
        (~Connection(m, c, u1))
    & (A" c1 : Comm_Object, u1 : Subject
        ((N"Active_Comm_Object(c1) <-> c1 ~=c & Active_Comm_Object(c1))
```

```
        & (N"Send_DAC_List(u1, c1)< -> c1 ~= c & Send_DAC_List(u1, c1))
        & (N"Receive_DAC_List(u1, c1) <-> c1 ~= c & Receive_DAC_List(u1, c1))
        & (N"Owner(u1, c1) <-> c1 ~= c & Owner(u1, c1))))
    |NC"( Active_Comm_Object, Send_DAC_List, Receive_DAC_List, Owner)

Transform Activate_Ext_Subject(u : Subject, l1 : Level, l2 : Level,
                                                    sm : Subject_Mode) External

Effect

    ~Active_Subject(u)
    & Lteq(l1, Max_Level(Resides_On(u)))
    & Lteq(Min_Level(Resides_On(u)), l2)
    & Lteq(l2, l1)
    & A" u1 : Subject
        ((N"Active_Subject(u1) <-> u1 = u | Active_Subject(u1))
        & (N"Subject_Mode(u1) = (u1 = u = >sm
                                        < > Subject_Mode(u1)))
        & (u1 = u => (Lteq(l2, N"Current_Level(u1))
                    & Lteq(N"Current_Level(u1), l1))
                < > (N"Current_Level(u1) = Current_Level(u1)))
        & (N"Max_Level(u1) = (u1 = u = > l1
                                        < >  Max_Level(u1)))
        & (N"Min_Level(u1) = (u1 = u = >l2
                                        < >  Min_Level(u1))))
    |NC"(Active_Subject, Subject_Mode, Current_Level, Max_Level, Min_Level)

Transform Deactivate_Ext_Subject(u : Subject) External

Effect

    A" c : Comm_Object, m : mode
    (~Connection(m, c, u) &~ Owner(u, c))
    & A" u1 : Subject,
    (N"Active_Subject(u1) < -> u1 ~= u & Active_Subject(u1))
     |NC"(Active_Subject)
```

B1 SECURITY FOR SPERRY 1100 OPERATING SYSTEM

R. E. Ashland
Executive Systems Development
P.O. Box 43942
St. Paul, MN. 55164-9907
(612) 635-6082

INTRODUCTION

This paper discusses how Sperry is enhancing the 1100 Operating System to meet the DoDCSC B1 security level. We consider this to be a major development feature. However, our B1 effort is not without limitations and is only an evolutionary step towards higher security levels.

Some of the discussions in this paper deal with Sperry 1100 OS unique concepts, and we recognize that you may find some of this not applicable to your environment. Hopefully, you will gain an overview of our B1 efforts.

Implementation Requirements

In addition to providing all the necessary features for a B1 system, we also have the following requirements:

1. Full compatibility for current users.

   a. Customers that choose not to use security cannot be impacted in any way.

   b. Customers that utilize security features can only be affected in the following ways:

      - system generation and site administrator/security officer procedures.

      - interactive terminal sign-on process.

   Existing programs cannot be impacted.

2. Performance must be maintained, where possible.

3. All user types must be included; that is, batch, demand (interactive), transaction (TIP), and MAPPER.

Scope of B1 Security

Adding security for transaction processing extends protection to a major type of system user. Security at the B1 level will not be provided for UNIX, Distributed Data Processing or networking.

Data Base Management (DMS 1100) security will not be multi-layered within one DMS. However, more than one DMS, each of a homogeneous data security level, may exist in one system. Each DMS copy is fully isolated and user access is fully controlled.

Current 1100 OS Security Status

Sperry received an unofficial C1 rating in 1983 on an older Executive level. We have corrected some of the noted deficiencies, have added discretionary file ownership to files, and by adding erasure of storage residue left from another user, we will have the requirements for a C2 system for batch and demand users.

Sperry's Security Strategy

We do not plan to have our system formally evaluated at C2, but we will complete the development of the B1 features discussed in this paper, continue our Preliminary Product Evaluation phase with DoDCSC, and move directly towards a formal B1 rating.

Brief 1100 OS overview

The 1100 OS is a complete, and, therefore, a large system. The major components for purposes of our security discussions are:

- EXEC - Controls hardware and all users; security kernel.
- CMS - Communications management.
- UDS/DMS - Data base management.
- MAPPER - Interactive information manager.
- TIP - Transaction processing.
- SIMAN - Site administrator's tool for Security.

SIMAN is used to establish the user security profile. The EXEC, along with CMS, TIP, and MAPPER, process user sign-on. All of these components share in enforcing the B1 mandatory labelling policy, and are therefore part of the TCB.

Customers tailor their selected system components to provide the type of system suited for their environment, such as banking, business, airlines, universities, government, or defense.

B1 FEATURES[1]

In addition to erasing storage residue, the following features will be added to the 1100 OS to achieve a B1 security level:

- Common sign-on for all users.
- Password controls.
- Compartment labelling of all objects.
- Transaction file and program security.
- Device security.
- Output labelling.

Common Sign-on

Batch and demand users currently have password and clearance level control. Transaction mode (call TIP, for Transaction Interface Package) and MAPPER will be placed under this same sign-on process. In addition, the following extensions will be made to the sign-on process:

- Hacker frustration, with terminal disabling, if selected by the Security Officer (SO).
- Password expiration - requiring a new password at the time interval selected by the Security Officer.
- Password encryption.
- Password phrasing capability by allowing lengths of 30 characters.
- Compartments - which categories of data the user is allowed and chooses to access.
- Project-id control. Sperry currently attaches a project name to files, which is used for file administration purposes. Control of which project-ids a user may use will be provided.
- Least Privilege - permit the selection of the desired privileges, within the set allowed, for this session.
- Session mode allowed - demand, MAPPER, or one of 9 TIP applications.

The Executive, along with SIMAN, the SO's administrative tool, will be enhanced to provide these features. In addition, the Log Analyzer (LA) will be enhanced to provide logging reports, for auditing purposes.

Compartments

A compartment set, identifying the category of data, along with the security sensitivity level, constitute the security attributes for B1 mandatory data labelling. Sensitivity level is currently used to label files in batch/demand environments. Compartments will be added to this label and extended to TIP files and messages, tape and disk units and volumes, and printers and printed output.

The following compartment features will be provided:

- SO control of creating, deleting and renaming the current set of compartments.
- User selection of sign-on, from allowed set, of compartments desired for this session.
- 900 possible compartments, with no added file I/O if only 30 compartments used.
- Versioning control to recognize changes in the current compartment set.

Transaction (TIP) Security

Four areas of Transaction Processing will be extended for B1 security:

1. TIP terminal sign-on. Actually a C2 requirement, this sign-on will be just like demand sign-on. Maintaining session information about users is a major change for TIP, in that currently no information is saved about a user from one transaction to the next transaction.
2. TIP file security. The Executive file containers for TIP files have security attributes which will be accelerated for TIP user access checks.
3. TIP program access checks. Transaction programs may be grouped and user access rights to these groups may be controlled, as well as which terminals may be used to transact these programs.
4. TIP message security. TIP terminals and TIP users will be checked against the TIP message label (compartment set and clearance level) for access rights.

TIP terminal connectivity and performance will not be sacrificed with these features.

Device Security and Output Labelling

The following will have security attributes:

- Removable disk units and packs.
- Tape units and tape reels.
- Printers and printed output.
- Terminals and terminal messages.

Device security attributes will be specified via SIMAN and validation at reference time performed by the Executive.

Output labelling will be written to all tape reels and removable disks packs by prep utilities or at tape writing.

Printed output labels will indicate the compartment set and the symbolic sensitivity level on printed output.

Auditing

Additional required auditing will be added and existing audits enhanced to the current system log for the new B1 features. Reports of security-related events will be produced upon request.

CONCLUSION

This has been a brief overview of what features Sperry is adding to the 1100 OS to provide a B1 system.

1. Lee, T.M.P., "Future Directions of Security for Sperry Series 1100 Computers", 7th DoD/NBS Computer Security Conference, Sept. 24-26, 1984.

DESIGNING THE GEMSOS SECURITY KERNEL FOR SECURITY AND PERFORMANCE

Dr. Roger R. Schell
Dr. Tien F. Tao
Mark Heckman

Gemini Computers, Incorporated
P. O. Box 222417
Carmel, California 93922

INTRODUCTION

Gemini Computers, Inc., offers as a commercial product a family of secure, high-performance computer systems based on the Intel iAPX 286 microprocessor. These systems are designed to meet the Class B3 requirements of the DoD Trusted Computer Computer System Evaluation Criteria,[1] and a developmental evaluation by the DoD Computer Security Center is ongoing. An earlier paper[2] of about a year ago discussed the major concepts underlying the design and the functionality of the Gemini Multiprocessing Secure Operating System (GEMSOS). The security kernel as discussed in that paper is structured into eleven distinct layers, and as of that time only the lower five layers were implemented. All of the layers described have now been implemented and delivered as a Version 0 kernel, and a production Version 1 is currently being implemented. The purpose of this paper is to report on the major design choices for security functionality and to report the results of initial system performance measurements on the Version 0 GEMSOS commercial product.

BACKGROUND

The GEMSOS security kernel design is based on the trusted computer system technology that has emerged over the past decade. This technology provides a high degree of assurance that a system developed in accordance with the principles underlying the DoD Trusted Computer System Evaluation Criteria can be objectively evaluated to determine its ability to protect sensitive information from unauthorized viewing or modification. The primary experience with this technology is with general purpose operating systems for single processor computers.

The Gemini design extends this security technology into the realm of a multiprocessor computer specifically intended for incorporation as a component of embedded systems. The question is often raised whether secure computers can be expected to deliver high performance. In the design of the GEMSOS security kernel, a good deal of attention has been given to supporting throughput and response time without adverse impact on security assurance. Of particular importance are the operating system techniques provided to ensure that the multiple processors can indeed be used effectively to increase overall system performance in concurrent computing applications.

The Reference Monitor Foundation

The reference monitor is the primary abstraction for dealing with a system that is designed to be "evaluable" with respect to security, i.e., a system for which we want to have a high degree of assurance of its correct security behavior. In this abstraction a system is considered as a set of active entities called "subjects" and a set of passive entities called "objects". The reference monitor is the abstraction for the control over the relationships between subjects and objects and for the manager of the physical resources of a system. To be effective in providing security, the implementation of a reference monitor must be: (1) tamper-proof, (2) always invoked, and (3) simple enough for analysis. The hardware and software that implement a reference monitor that meets these principles is defined as a security kernel.[3]

Security Policy Model

For a specific set of applications, e.g., for DoD systems, there will be a particularization of the reference monitor abstraction that incorporates the "security policy" (the desired security behavior) of the system. This particularization is formally defined in a "security policy model." The choice of a model will be influenced by a desire to have an intuitive tie to the engineering properties of the target system. Thus, in selecting a model for a trusted computer, it is desirable that the model's objects can easily represent the security-relevant information repositories of the contemplated applications. By far, the most widely used formal security policy model is the Bell-LaPadula model.[4] This model has a level of abstraction that is high enough to permit application to a wide variety of specific designs and is also deliberately designed to be extended to support system-specific policy refinements. This model has been used as the basis of the GEMSOS security kernel design.

An Extensible TCB

A pivotal concept in developing a system that is secure in a practical sense is the identification of a Trusted Computing Base (TCB). A security policy model is expected to model a broad range of actual systems. However, if the system is of a practical size it is expected that it will be too complex to systematically evaluate for security. However, the reference monitor concept provides a basis for identifying a small and simple subset of the system that is responsible for and able to assure the security of the total system. This subset is called the TCB. The TCB includes both (1) the security kernel that implements the reference monitor and manages the physical resources, and (2) the "trusted subjects" that support refinements to the fundamental policy supported by the security kernel.

As noted in a recent paper by Marvin Schaefer of the DoD Computer Security Center,[5] if the TCB has a strict hierarchical layering it is possible to extend a mandatory policy security kernel to support a richer set of security properties, such as those desired for a discretionary security policy of a particular application. The GEMSOS security kernel has the kind of strict layering that was postulated. The Intel iAPX 286 processor that is used in the Gemini computer provides four strictly hierarchical, hardware enforced protection rings that enforce the strict layering. In particular, the most privileged ring (Ring 0) is devoted to the mandatory policy security kernel. Typically, Ring 1 would be similarly devoted to the particular discretionary policy of an application.

The key to the evaluation of the security of a TCB is the Descriptive Top Level Specification (DTLS) and for the Class A1 a corresponding Formal Top Level Specification (FTLS) for the interface to the TCB. For the GEMSOS security kernel the approach is to have specifications that are themselves layered so that there is a DTLS of the mandatory security kernel as well as a DTLS for the refinements, such as that for discretionary security policy; the latter specification includes references to the underlying mandatory DTLS. Thus the GEMSOS security kernel provides an ideal foundation for a truly extensible TCB for support to a wide range of extended policies, such as those frequently encountered in military embedded systems.

MAJOR SECURITY CONCEPTS

A reference monitor implementation such as the GEMSOS security kernel must mediate all access by active entities (called "subjects") to passive entities (called "objects"). The GEMSOS kernel permits or prevents each access by a subject to an object based on relationships between the subject's authorization and the objects sensitivity. This set of relationships is called a security policy. The GEMSOS mandatory policy security kernel can enforce any of the various policies that are represented by the lattice of security labels used in the Bell and LaPadula mathematical security model.[4] This model provides a set of rules for controlling the dissemination and modification of information in a secure system. Kernel calls map into the rules of the model, and internal data structures represent the model's mathematical sets. The GEMSOS security kernel is secure because it is a valid interpretation of this security model. In the sections that follow, we define and introduce the major security system concepts and then use those concepts to describe the specific GEMSOS security kernel features.

Subjects and Objects

Rules in the Bell and LaPadula model are concerned with controlling the access of subjects to objects. Subjects are "active" entities that observe or modify objects. Objects are "passive" entities that are observed or modified. A subject is defined as a process executing in a specific domain and may be thought of as a (process, domain) pair. Objects are distinct, logical entities that contain information and possess security attributes called access classes.

Domains

A domain is a set of objects to which a subject has a given type of access, e.g., the "observe domain." Domains in the GEMSOS security kernel are determined by the hardware-enforced ring mechanism. The rings define a set of hierarchically ordered domains (see the section on ring integrity).

Security Policy

The permission of "authorized" access and the prevention of "unauthorized" access by subjects to objects constitutes the security enforced by a system. A security policy is the set of relations between subjects' authorizations and objects' sensitivities that determines permissible access. A system that enforces a particular security policy may be said to be secure only with respect to that policy.[3]

Nondiscretionary Security. A security policy based on externally defined constraints enforced by a secure system is called a "mandatory" or nondiscretionary security policy. A nondiscretionary policy controls all accesses by subjects to objects and may never be modified or bypassed within the system. The military classification and compartment policy is an example of a nondiscretionary policy.

Discretionary Security. Within the limits of mandatory controls, authorized subjects in the system may place additional constraints on other subjects' access to objects. This internally modifiable set of constraints is called a discretionary security policy. The military "need-to-know" policy is an example of a discretionary policy. The complete security of a system may include both mandatory and discretionary controls, but while discretionary controls provide finer access "granularity," in no case may they override mandatory controls. The GEMSOS mandatory policy security kernel (viz., Ring 0) enforces a nondiscretionary security policy and provides a base to which a discretionary policy may be added (e.g., in the Ring 1 "supervisor" domain, as noted above).

Access Classes

Every entity in the GEMSOS security kernel possesses an access class. The access class of an object reflects the object's sensitivity, viz., its "classification." The access class of a subject reflects the subject's authorizations to observe and modify objects, viz., its "clearance." The complete nondiscretionary (mandatory) security attributes of any subject or object in the GEMSOS security kernel are defined by its access class.

Security Labels. Security labels are attached to every entity (subject and object) in the system. A security label is a representation of an entity's access class.

Access Components. An access component is a way of describing separately the secrecy and integrity security attributes that consti-

tute an access class. Discussion of access components simplifies the description of relations between subjects and objects based on access classes (i.e., the non-discretionary security policy). In the GEMSOS security kernel, an access component is defined similarly to a Bell and LaPadula "security level." Both the secrecy and integrity access components consists of two parts: a hierarchical classification level and a set of compartments (induced by disjoint "categories").

Access Component Dominance. One access component (A1) is said to dominate another access component (A2) if the hierarchical level of A1 is greater than or equal to that of A2, and A1's compartments are a superset of A2's compartments. The symbol "}=" is used in following sections to indicate dominance (e.g., "A1 dominates A2" is depicted as "A1 }= A2")

Dissemination and Modification of Information

The nondiscretionary security policy enforced by the GEMSOS security kernel addresses both the secure dissemination and secure modification of information. The access class of every subject and object in the GEMSOS security kernel contains an access component to control dissemination and another access component to control modification. These two access components are referred to as "secrecy" and "integrity," respectively.

Secrecy Protection

Secrecy protection in the GEMSOS security kernel is similar to the usual interpretations of "security" in the Bell and LaPadula model.[4] The notion of secrecy is concerned with the secure distribution of information. Although the exact statement of security in the model is considerably more complex, the rules for enforcing secrecy protection in the GEMSOS security kernel can be simply described by two properties:

1) If a subject has "observe" access to an object, the secrecy access component of the subject must dominate the secrecy access component of the object.

2) If a subject has "modify" access to an object, the secrecy access component of the object must dominate the secrecy access component of the subject.

Property 1 is called the "no-read-up" or "simple security" property. Its effect is to keep low-secrecy subjects from observing information of higher secrecy. Property 2 is called the "no-write-down" property or "*-property" (read "star-property"). Its purpose is to keep high-secrecy subjects from improperly transmitting sensitive information to low-secrecy subjects (e.g., with a Trojan Horse). Thus, a low-secrecy subject can never directly or indirectly observe high-secrecy information.

Integrity

The concept of integrity is concerned with the secure modification of information. The GEMSOS security kernel provides two complementary mechanisms for enforcing integrity: one in software and one in hardware. The software mechanism enforces an integrity policy equivalent to the strict integrity policy described by Biba.[6] The hardware mechanism supports Multics-like hierarchical protection rings[7]. The integrity enforced by the ring mechanism is equivalent to the notion of "program integrity"[8] and is a subset of the strict integrity policy.

The two mechanisms for enforcing integrity are provided for efficiency reasons. Were no ring mechanism available, the complete strict integrity policy could still be enforced by separate processes (rather than separate rings) using the software mechanism, but the hardware ring switching mechanism is considerably faster than process switching.

Strict Integrity. Like secrecy protection, the rules for enforcing strict integrity protection in the GEMSOS security kernel can be described by two properties:

1) If a subject has "modify" access to an object then the integrity access component of the subject dominates the integrity access component of the object.

2) If a subject has "observe" access to an object then the integrity access component of the object dominates the integrity access component of the subject.

Property 1 is called the "simple integrity" property. Its purpose is to prevent subjects of low integrity from modifying objects of higher integrity. Property 2 is called the "integrity *-property." It prevents high-integrity subjects from observing and relying on information that a low-integrity subject might have modified. If high-integrity subjects could observe low integrity information then their behavior might be improperly influenced ("spoofed") by a low-integrity subject. The two integrity properties prevent low-integrity subjects from directly and indirectly modifying high-integrity information.

Ring Integrity. Under the GEMSOS ring integrity mechanism, each subject and object possesses a hierarchical ring level ranging from 1 (most privileged) to 3 (least privileged). Subjects are only permitted access (observe, modify, or both) to objects with equal or greater ring numbers (equal or less privileged rings). No access to objects with a lower ring number is permitted.

For example, only the GEMSOS security kernel supervisor is allowed to have ring level 1. No application program, therefore, can access any part of the supervisor, but the supervisor can access any object in rings 1 thru 3. Ring 0 of a Gemini system is reserved for the isolation and protection of the security kernel.

Trusted Subjects

In general, the properties of secrecy and integrity are strictly enforced by the GEMSOS security kernel. Rigid adherence to these properties, however, can complicate the use of a system by forcing extremely fine "granularity" of security on objects. For example, imagine a system where storage objects are

large files, and which has an incoming stream of messages at different access classes. One logical method of distributing messages would be to have all incoming messages put in the same file, and to have a single process distribute the messages to the proper recipients.

The secrecy and integrity properties described above, however, force a secure system to create a different file for each access class in which to store messages. In addition, a different process of the appropriate class must be used to distribute messages out of each file. Distributing messages, a relatively simple operation on most systems, could become a needlessly complicated, resource-consuming procedure in a secure system.

Imagine instead that the secrecy and integrity *-properties were relaxed just for this application. All incoming messages could be temporarily put in the same file and a single process could distribute messages of any access class. What would be the security characteristics of the file and process? The file would have the most sensitive access class possible, since it could conceivably contain messages of that class. Messages in the file, however, could be of lower classes. The distributing process would need authorization to observe objects of the highest class in order to read the file, and be able to modify objects from the most sensitive class down to the least sensitive in order to distribute the messages at their appropriate class. Since the process would be operating at many different access classes simultaneously, it must be trusted not to improperly pass sensitive messages to subjects with insufficient authorizations.

In order to support this type of application, the Bell and LaPadula model includes a "trusted subject" as part of the model. Trusted subjects in the model are subjects unconstrained by the *-property. Trusted subjects in the GEMSOS security kernel, unlike Bell and LaPadula's definition, are trusted (the *-property is relaxed) only within a given range, and are therefore "multilevel" subjects instead of general trusted subjects. Even with this additional security constraint the GEMSOS security kernel is still a valid interpretation of the Bell and LaPadula model for, as Bell and LaPadula write, "... restrictions of the concept of security will not require reproof of the properties already established because additional restrictions can only reduce the set of reachable states."[4]

GEMSOS SECURITY KERNEL FEATURES

Each of the structures and concepts described above is embodied in some way in the GEMSOS security kernel. The GEMSOS security kernel organization for implementation has been described previously,[2] and the reader interested in the specific primitives and kernel calls identified in the following discussions is encouraged to review this description. The following sections describe the major security features. The basic abstraction used in the GEMSOS design are segments, processes and devices. The segments are instances of "objects" of the model. The processes and devices are "subjects" of the model.

Security Labels

Security labels are representations of the sensitivity of objects and the authorizations of subjects. A security label is attached to every subject and object in the GEMSOS security kernel. Security labels are called "access classes" since they symbolize the complete set of security attributes possessed by each entity. The GEMSOS security kernel access classes (security labels) are records with two fields, representing secrecy and integrity access components. For secrecy the default label has eight (8) hierarchical classifications and twenty-nine (29) non-hierarchical categories. For strict integrity the default label has eight (8) hierarchical classifications and sixteen (16) non-hierarchical categories.

Some entities in the GEMSOS security kernel have two access classes: a maximum and a minimum. The secrecy and integrity access components of the maximum access class always dominate the secrecy and integrity access components of the minimum access class. The maximum and minimum classes together describe a range of permissible access classes.

Segments

All information in a Gemini system is contained in discreet, logical objects called segments. Each segment has an access class that reflects the sensitivity of information contained in the segment. Segments may be simultaneously and independently shared by multiple subjects but access to the segment (observe, modify, or both) on the part of each subject is controlled by the relationship between the segment's access class and each subject's access class, in accordance with the security properties of the model.

Every segment in the GEMSOS security kernel has a unique identifier. This identifier is effectively different for every segment ever created in any Gemini system. Unique identifiers are used internal to the kernel to prevent "spoofing" of the system by substituting one segment for another and are not visible at the kernel interface.

Eventcounts and Sequencers. The GEMSOS security kernel uses abstract data objects called "eventcounts" and "sequencers" for process synchronization and communication. These objects may be observed and modified by subjects so, to preserve security, the GEMSOS security kernel must control access to them. In order to identify eventcounts and sequencers, one eventcount and one sequencer are associated with the name of each segment. Each segment name, therefore, is used to identify an eventcount and a sequencer as well as a segment (see the section below on segment aliasing for more information on segment names). The eventcount and sequencer associated with a segment name have the same access class as the segment whose name they share. Process synchronization and communication are thus subject to the same rules of security as observing and modifying segments.

In order to modify an eventcount (using the primitive "advance") a process must be

permitted modify access to the segment. Similarly, in order to observe an eventcount (using the primitives "read" or "await") a process must be permitted observe access to the segment. The primitive "ticket" for sequencers requires the potential for both observe and modify access to the segment. See the description by Reed[9] for more information on eventcounts and sequencers.

Volumes. Secondary storage in Gemini systems is divided into distinct logical volumes. Each segment is associated with only one volume, determined when the segment is created. Each volume may be considered to be a collection of segments. Volumes have two access classes, a maximum and a minimum, assigned when the volume is formatted. The secrecy and integrity components of the maximum access class must dominate the secrecy and integrity of the minimum class. The maximum and minimum volume access classes are upper and lower limits on the security of information contained on the volume (see the section below on the use of volumes).

Processes

A subject in the GEMSOS security kernel is a (process, domain) pair, where the domain is determined by the current hierarchical ring level at which the process is executing. The ring level determines the set of objects to which, within security constraints, the process potentially has access. Processes may change their ring levels, but the same process executing in a different ring is a different subject. The primary application of rings envisioned under the GEMSOS security kernel is for the creation of distinct domains so that a process can have up to three (rings 1, 2, and 3) "subjects." Each subject has a minimum and a maximum access class (security label) that is typically uniform for a process, no matter which ring it executes in. The secrecy and integrity access components of the maximum access class dominate those of the minimum access class. If the maximum and minimum access classes are equal then the subject is a "single-level" subject. If the two classes are unequal then the subject is a "multilevel" subject.

Single-level Subjects. Subjects that have equal maximum and minimum access classes are single-level subjects. Single-level subjects may only have the access to objects permitted by the simple and "*" secrecy and integrity properties (see the sections on secrecy and integrity).

Multilevel Subjects. Multilevel subjects have unequal maximum and minimum access classes. This property of multilevel subjects gives them the ability to have both observe and modify access to objects whose access classes fall between the subject's minimum and maximum. Multilevel subjects are the the GEMSOS security kernel implementation of "trusted subjects" (see the section on trusted subjects). Unlike general trusted subjects, however, multilevel subjects are only trusted within a range demarcated by their maximum and minimum access classes. Within this range, multilevel subjects are not constrained by the *-properties of secrecy and integrity (but they are still subject to ring integrity). Only subjects guaranteed not to improperly downgrade or modify information should be created as multilevel subjects.

I/O Devices

I/O devices are viewed by processes as system processes. Processes communicate with I/O devices using shared segments and may also use eventcounts. Like other processes, I/O devices have both maximum and minimum access classes. These limits are intended to reflect the security constraints imposed by the physical environments in which devices are located and are critical to the employment of a secure computer in a multilevel environment.

Devices may be either single-level or multilevel. Unlike processes, which are classified single-level or multilevel based on their minimum and maximum access classes, the Criteria[1] categorizes I/O devices as "single-level" or "multilevel" based on the access classes of the data they manipulate. Data transmitted or received by a single-level device has no attached security label. Single-level devices thus consider all data to have a single access class. Data transmitted or received by a multilevel device has a security label attached to or stored with the data in the same form as the data. Multilevel devices therefore may handle data with a range of access classes.

Single-level Devices. A single-level device handles data to which no explicit security label is attached. In many environments the minimum and maximum access classes for a single-level device will be the same. A single-level device at a given time treats all input data as having a single access class, which is determined by the security of the physical environment at that time. Output data must have an access class that falls within the range of the device's maximum and minimum access classes.

In order to establish communication between a process and a single-level device in the GEMSOS security kernel (called "attaching" the device), the range of the subject must "intersect" the range of the device. Specifically, the following relationships between the process's access classes and the device's access classes must hold to ensure that the process will be able to receive or send data through the device.

1) To receive ("read") information:

Process maximum secrecy }=
Device minimum secrecy

Device maximum integrity }=
Process minimum integrity

2) To send ("write") information:

Device maximum secrecy }=
Process minimum secrecy

Process maximum integrity }=
Device minimum integrity

An example of a single-level device with different minimum and maximum access classes is a log-on terminal in a room to which users

with authorizations ranging from the highest possible (system-high) access class down to the lowest possible (system-low) access class have access. In this example, the maximum and minimum access classes of the device would be system-high and system-low respectively, although narrower ranges are possible in other situations. There is only the single terminal in the room and only one person is allowed in the room at a time. Users log on to the system through a "trusted path"[1] which allows the GEMSOS security kernel to directly and securely determine their access class; it then creates a process of the same access class to represent the user in the system.

After a user has logged on using the trusted path, and the user's process has attached the device, the GEMSOS security kernel considers the security of the terminal device environment to be the same as the security of the user's process. Different users will have different access classes, but at a given time there is only one user so the data has only a single access class.

When a single-level device receives data, an access class (security label) must be established for the data. If the current security of the device has been reliably transmitted to the GEMSOS security kernel (e.g., through a trusted path) then the attached process will have an access class that represents the current security of the device. The received data is usually assigned the maximum secrecy and minimum integrity of the process that attached the device. A single-level device with a range of access classes, as can be seen from this example, must have a trusted path of some sort in order to be used in a secure manner. If not, then the minimum and maximum access classes of the single-level device should be identical.

Multi-level Devices. Any data input or output through a multilevel device must have an access class that falls within the range defined by the device's maximum and minimum access classes. Multilevel devices may handle data with a range of access classes. All data transmitted or received by a multilevel device has a security label attached or stored along with the data.

In order for a process to attach (establish communication with) a multilevel I/O device, the following relationships between the process's access classes and the device's access classes must hold to ensure that the process can send and receive information through the device without violating security.

1) To receive ("read") information:

Process maximum secrecy }=
Device maximum secrecy

Device minimum integrity }=
Process minimum integrity

2) To send ("write") information:

Device minimum secrecy }=
Process minimum secrecy

Process maximum integrity }=
Device maximum Integrity

Process and Segment Interaction

This section explains how the GEMSOS security kernel controls the ability of subjects, viz., (process, domain) pairs, to access objects (segments). A process may create and destroy segments, may add segments to and delete segments from the process's address space, and may move segments between main and secondary storage.

The total set of objects to which a subject potentially has access is the subject's access domain. A subject's access domain is determined by the subject's hierarchical ring level. The subset of the access domain that includes all objects to which, at a given time, a subject actually has access is called the subject's address space. Segments are brought into a subject's address space using the kernel call "makeknown_segment".

Access Modes. The GEMSOS security kernel allows processes to have execute only, read-execute, read only, and read-write access mode combinations to segments. Of these access mode combinations, all but read-write are considered to be "observe" type access modes. Read-write is both an "observe" type and a "modify" type access mode. The GEMSOS security kernel has no write only (modify only type) access mode for segments.

A process may have only one access mode combination to a segment at a time, but may simultaneously have different access modes to different segments. The access mode a process has to a segment is selected by the process at the time the segment is brought into the subject's address space. The actual type of access selected need only be a subset of the potential types of access allowed by the security policy. For example, if a process may potentially have both observe and modify type access to a segment based on the relationship between the process's and segment's access classes, it need not select the read-write access mode (observe and modify) to the segment, but can instead select any of the observe type access modes, since "observe" is a subset of its potential access types.

In order to have any of the observe access mode combinations to a segment, a process's maximum secrecy access component must dominate the segment's secrecy and the segment's integrity access component must dominate the process's minimum integrity. These requirements enforce the simple security and integrity "*" properties. In order to have modify access to a segment, the segment's secrecy access component must dominate the process's minimum secrecy and the process's maximum integrity access component must dominate the segment's integrity. These requirements enforce the secrecy "*" and simple integrity properties. In order to have both observe and modify access to a segment all four of these requirements must be met. Multilevel processes thus potentially have both observe and modify access to any segment whose access class falls within the process's range, while single-level processes are only permitted both observe and modify access to segments with the same access class as the process.

Segment Naming (Aliasing). In order to create or delete a segment, or to add a segment to its address space, a process must tell the GEMSOS security kernel the process-local name of the segment. The method of naming segments is called aliasing. Aliasing allows processes to uniquely identify shared segments in the system while still maintaining security.

A segment name consists of a system-wide component and a process-local component. The system-wide component is an index called an "entry number." A segment's entry number is the same for all processes, can be stored for future reference to the segment, and can be passed to and used by other processes. Were it not for the danger of covert information channels, the totally "flat" (non-hierarchical) entry number naming scheme would itself be sufficient for naming all segments.

Due to the danger of covert channels however, the system-wide entry number of a segment is always relative to a "mentor" segment. The mentor segment is identified by a process-local number that is not unique across processes, that cannot meaningfully be passed to another process, and that cannot be saved for future use (although the mentor itself also has a system-wide name, as described below). The paired process-local mentor segment number and system-wide entry number constitute a process-local segment alias used by a process for identifying the segment to the kernel.

A segment's alias is different from its ("invisible") unique identifier (described in the section on segments). Only one segment can have a particular (mentor, entry) pair as its name at a time, but if that segment is deleted then another segment can be created with the same name. The unique identifier of the new segment, however, will be different from the old segment.

A mentor segment may not be deleted until all segments named with that mentor have been deleted. This restriction prevents the problem of "zombie" segments that, although they exist in storage, cannot be accessed or deleted since they cannot be named.

Segment Naming Hierarchy. The process-local segment alias, viz., (mentor, entry) pair, of a segment can be used to add the segment to a process's address space. When a segment is entered into a process's address space, the process assigns the segment its own process-local segment number. The segment may then itself be used as a mentor segment if its process-local segment number is used as the mentor in the alias of another segment. Recursive application of this naming scheme results in a "hierarchy" of segment aliases.

Compatibility Property. The hierarchical segment naming scheme, using segment aliases, allows the GEMSOS security kernel to preserve system security by preventing the creation of covert channels through the use of segment names. Were processes able to name and thereby sense the presence or absence of any segment in the system (as they would in a flat naming scheme), this would constitute a source of covert channels. High secrecy level processes could signal low secrecy level processes, and low integrity level processes could signal high integrity level processes, just by creating and deleting segments.

The GEMSOS security kernel alias hierarchy, however, allows processes to name segments only where that naming will not cause a covert channel. The hierarchy prevents covert channels by strictly ordering the security relationship between segments and their mentors:

Segment Secrecy }= Mentor Secrecy
Mentor Integrity }= Segment Integrity

These properties of the hierarchy have been called the Compatibility Property[4] (for secrecy) and the Inverse Compatibility Property (for integrity).[6]

The compatibility and inverse compatibility properties are maintained because of the way segments are named. A segment name consists of a (mentor, entry) pair. The entry number is relative to that particular mentor and is considered to be information associated with the mentor segment. A segment's entry number, as a result, is considered to have the same access class as the segment's mentor segment.

Whenever a process tries to create or delete a segment, or to add a segment to its address space, the process must "observe" the segment's entry number to see if, in fact, the segment exists. Since entry numbers are information associated with a mentor, a process must potentially have observe access to the segment's mentor in order to name the segment.

A mentor's alias consists of another (mentor, entry) pair, its mentor is named in the same fashion, and so on recursively back to the "root" of the naming hierarchy. Each segment's alias can thus be thought of as a vector consisting of a series of entry numbers that together uniquely describe a "path" to the segment. Clearly, in order to name a segment, a process must be able to name every mentor segment in the segment's path and it must therefore potentially have observe access to each of the mentor segments.

If the compatibility property was not maintained and a segment's compromise did not necessarily dominate the secrecy of its mentor, a situation might arise where a process that should have access to the segment based on the relationships between their access classes cannot have it, since the process cannot get observe access to the mentor. Every segment's secrecy, therefore, must dominate the secrecy of its mentor. Applying this rule recursively back to the root gives the result that secrecy is monotonically nondecreasing following a name path from the root to any segment.

Every segment has a unique path. This is achieved by requiring all segments with the same mentor to have unique entry numbers. Whenever a process creates or deletes a segment it specifies the name of the segment (conceptually "modifying" the segment's entry number to indicate the entry is in use or free). Since entry numbers are information

associated with a mentor, a process must potentially have modify access to the mentor in order to create or delete a segment's name.

Because a segment name is a vector consisting of a series of entry numbers, changing any of the constituent entries would change the segment's name. A segment's name, therefore, must have at least the integrity of the segment (i.e., must dominate the segment's integrity). Since a segment's name consists of a (mentor, entry) pair, and since the entry number has the same access class as the mentor segment, the integrity of the mentor segment must dominate the integrity of the segment. Applying this rule recursively back to the root gives the result that integrity is monotonically non-increasing following a name path from the root to any segment.

To create a segment, a process requires the potential for both observe and modify access to the segment's mentor segment. Creating a segment affects only the segment's name, which is associated with the mentor, and does not affect the contents of, or information associated with, the segment being created. A process that creates a segment, therefore, does not need to be able to access the segment it creates.

To delete a segment a process requires the potential for both observe and modify access to the segment's mentor and, in addition, requires the potential for observe access to the segment being deleted. This additional restriction is necessary because mentor segments may not be deleted. A process deleting a segment must therefore know if the segment it is trying to delete is a mentor segment, requiring the process to "observe" if any other segments are named using the segment as a mentor. Those segments names, as stated above, are information associated with the mentor and possess the same access class.

The Use of Volumes. Volumes are collections of segments useful for physically organizing and protecting information of similar access classes. Volumes are brought into the GEMSOS security kernel using the kernel call "mount_volume." The first time a volume is mounted, it is uniquely associated for the life of that volume with a segment that will serve as the "root" mentor to segments on the volume. The segment used as a volume mentor may serve as a volume mentor to that volume only, and must not ever have been a mentor to other segments before the initial mounting of the volume. In this way all segments on the volume are guaranteed to have unique pathnames, distinct from the pathnames of segments on other volumes. Were pathnames indistinct, the system could be spoofed by substituting one segment for another of the same name.

Volumes may be unmounted and mounted repeatedly, but, to ensure unique pathnames, if its mentor is deleted a volume may never be remounted. A volume whose mentor is deleted must be reformatted to be reused -- a process that destroys whatever information the volume might contain. A volume mentor segment may not be deleted while the volume is mounted. A volume may not be unmounted if any of the segments on the volume are currently "known" in any process's address space. These precautions prevent the problem of "zombie" volumes (mounted volumes that cannot be unmounted because they are not addressable) and "orphan" segments (segments addressable but not able to be swapped-out to disk).

The names of all segments on a volume must have access classes that fall within the volume limits. Since some segments on the volume will have the volume mentor segment as their mentor, the volume mentor segment's access class must also fall within the maximum and minimum access classes of the volume. According to the compatibility property, secrecy is monotonically nondecreasing and integrity is monotonically nonincreasing. When a volume is first mounted, therefore, the maximum potential secrecy access component of segments on the volume is the volume maximum, but the effective minimum compromise access component becomes the mentor's secrecy. Similarly, the minimum possible integrity access component of segments on the volume is the volume minimum, but the effective maximum possible integrity becomes the the mentor's integrity.

In order to satisfy the compatibility property, the secrecy access component of any segment created on a volume must dominate the secrecy access component of the volume mentor and be dominated by the maximum compromise of the volume. The integrity access component of any segment created on the volume must dominate the minimum integrity of the volume and be dominated by the mentor's integrity. These relationships are summarized below (where S indicates Secrecy, and I indicates Integrity):

```
S(volume max) }=
   S(segment) }=
      S(volume mentor segment)

I(volume mentor segment) }=
   I(segment) }=
      I(volume min)
```

To prevent a covert channel caused by the mounting and unmounting of volumes, a process that mounts or unmounts a volume must have a minimum secrecy access component dominated by the secrecy of the volume mentor and a maximum secrecy access component that dominates the maximum compromise of the volume. The minimum integrity access component of the process must be dominated by the minimum integrity of the volume and the maximum integrity access component of the process must dominate the integrity of the volume mentor segment. That is:

```
S(process max) }=
   S(volume max) }=
      S(volume mentor segment) }=
         S(process min)

I(process max) }=
   I(volume mentor segment) }=
      I(volume min) }=
         I(process min)
```

An example of the use of volumes is a floppy diskette environment. Each floppy diskette is a different volume with its own maximum and minimum access classes. Disks are labeled with these classes at the time they are formatted. The first time a diskette is used, the process that represents the diskette

user in the system creates a mentor segment for the diskette volume and mounts the volume. Other users' processes may share the volume if the security policy allows them to "make known" the volume mentor segment in their address spaces. After the volume is unmounted and the diskette put away, the diskette may be reused so long as its mentor segment is not deleted. If the diskette volume's mentor is deleted, the diskette must be reformatted to be reused, destroying any information contained on the diskette.

SYSTEM PERFORMANCE MEASUREMENTS

The security-kernel approach to the design of a multilevel secure computer system offers a solution to the size and complexity problems that have dogged other approaches. However, some previous implementations of security kernels have resulted in systems with discouraging performance, reportedly as much as 75 to 90 percent below that of equivalent non-trusted systems.

Design Factors

After about a decade of work in the area of trusted systems, substantial information is available on factors that relate to some of the disappointing performance that has been experienced. Several of these factors are discussed below.

Language Efficiency. For verification purposes, security kernels are written in high-level languages which are chosen for features such as strong typing of data. Some of these languages tend to produce inefficient code. In the case of the GEMSOS security kernel, the PASCAL language has been chosen because of its support for evaluation. The compiler used is not particularly efficient, but is considered typical for microcomputer compilers. Thus there is some performance impact from choosing to use a higher-order language, but no significant additional impact from the choice to support security.

Hardware Support. The different security classes of users and information must be distinguished and where incompatible, kept separate; when hardware support is inadequate, the supporting overhead restricts the bandwidth of information that a secure computer system can process. Previous work has identified[3] four general architectural areas where hardware features are particularly useful or necessary: process management and switching, memory segmentation, Input/Output mediation, and execution domains. The Intel iAPX 286 processor used for the Gemini computers provides a high level of hardware support in all these areas.

System Architecture. The implementation choices for organizing the internal structure and the environment for applications have a major impact on the performance of any operating system. Generally the approach in the GEMSOS security kernel has been to take advantage of the techniques found effective in the industry as long as these do not adversely impact security. For example the choice of system computational model, e.g., process oriented or capability based, can directly affect system response time. We have chosen the proven process oriented approach, as reflected in the security discussion above. Two other choices of particular importance relate to the how multiprocessing is implemented, viz., techniques for avoiding bus contention and for preventing the kernel from being a critical section bottleneck. These are discussed below.

Bus contention is a potential performance concern in the Gemini multiprocessor configuration, since all processors share a single bus. In reality however, only shared, writable segments need be in a global memory on the shared bus. All other segments can be in processor-local memory. Our use of a purely virtual, segmented memory permits the kernel to determine exactly which are the shared, writable segments. The memory manager layer internal to the kernel totally controls the allocation to global memory to insure that only the required segments are in global memory. This policy can require some transfer between local and global memory but this structure markedly controls bus contention by allocating segments to the processor-local memory whenever possible. Our experience with sample applications is encouraging in that typically much less than 10% of the references of a processor are to a global memory. Thus, a number of processors can be effectively used on a single, shared bus.

In most, if not all, previous security kernel implementations, the kernel is a single critical section. This means that the kernel can be executed by only a single process at a time, and in addition cannot be interrupted. For a single processor, the adverse impact of this choice is somewhat contained in that there are no other processors that can be forced to wait. However, even in the single processor case real time response may be affected because there will be no response to an interrupt from an external device until any call to the kernel, that has begun before the interrupt, is completed. The GEMSOS security kernel is designed to be close to interruptible throughout its entire execution.

The impact of the critical section design choice is much more severe in the multiprocessor case. With a critical section, if one processor is executing in the kernel when another processor wants to invoke the kernel, the second processor must wait in essentially an "idle" condition until the first processor completes its execution of the kernel call. The degradation is clearly a function of the amount of service, viz., the number and type of calls, that the application demands of the kernel. In addition, the degradation increases as the number of processors increase. In the GEMSOS security kernel there are limited critical sections internal to the kernel itself that can result in contention, but the kernel itself is not a critical section so that multiple processors can execute simultaneously in the kernel.

Performance Results

Although the design approach to provide good performance is of interest, the real proof of any system is the actual measured performance. We have taken some preliminary measurements that focus on demonstrating (1)

the throughput performance for multiprocessor configurations and (2) the response to real-time inputs. It is emphasized that these measurements were taken on Version 0, and that several performance enhancements have been designed for Version 1 that have not yet been implemented. Although we believe the results illustrate the general behavior, these early measurements should not be considered a definitive characterization of the Gemini product.

Multiprocessing Throughput. We have prepared a message processing emulation that runs on the multiprocessor environment. The emulation is reminiscent of a military communications processing application. The processing consists of a "front end processor" servicing multiple communication lines, and interfacing to an additional single communication line. The demonstration is not connected to physical communication lines, but emulates receiving messages from an input buffer segment and putting output messages in an output buffer segment.

The details of the demonstration are not very important to the measurements, but will be briefly summarized. Messages are treated as a series of line blocks of 84 bytes each. Each message requires some amount of processing and then a resulting message is placed in the message queue segment for "transmission" through the output buffer segment. A message processing process is created for each pair of input communication lines; this process does all the message processing and places the message, a line block at a time, in the message queue. In addition there is a single output process that takes messages, a line block at a time, and puts them in the output buffer. The kernel synchronization primitives for eventcounts are used to ensure that each message processing process waits for room in the message queue and to ensure that the output process takes the messages from the queue when they are available.

All the processes and buffer segments are actually created and used. The only actual Input/Output is to a screen for interface to the test operator. A display is generated for each message line block, and at the end of the "test run" timing information based on the internal real-time clock is displayed. The demonstration is intended to show the additional capacity that can be provided by additional processors. A distinct "test run" is used for each processor configuration of from one to seven processors, emulating service for from two to fourteen input lines, with each processor servicing two lines. The test operator selects a parameter that controls the amount of processing for each message. This parameter determines the amount of processing that is in the demonstration application versus the amount of processing in the kernel. The processing is simulated by repeated execution of a mix of instructions taken from a communication processing application.

An estimate is made of the percentage of the total processing time spent in the application for various choices for the parameter that controls the amount of processing for each message. The measurements taken on a single processor are used to normalize the results for additional processors, so that the number of "effective processors" can be determined. The number of effective processors becomes the primary figure of merit for the true effectiveness of the multiprocessing. Because of the multiprocessor contention within the kernel, this will be reduced if the application requires extensive services from the kernel, viz., as the percentage of application processing is reduced.

The results of a series of actual measurements as described above are summarized in Figure 1. This shows that for substantial application processing, there is a nearly linear increase in system throughput as the number of processors is increased. This clearly reflects that there is very little contention between processors for the shared bus. Futhermore, even when only 85% of the processing is in the application, there is still effectively six processors worth of throughput for a seven processor configuration.

Real Time Response. We have prepared a set of tests that require real-time response to external input. For this test we process communications input in a way that requires character-at-a-time processing. For each character there is an interrupt and the system must respond before two additional characters are received, or else with the hardware used for the interface, the communication will be broken. This is not necessarily the preferred implementation for such communication, but serves as a useful test implementation. Thus if the interrupts occur frequently (i.e., for a high transmission rate), it is essential that the kernel be interruptible. The specific test is an implementation of the HDLC support for the X.25 protocol that is used for the Defense Data Network interface to a computer host. The test has no higher level "flow control" protocol, so the input data must be received in real time.

The communication is synchronous, so that the actual amount of time available to respond to the interrupt depends on the transmission rate used. The other primary parameter for the test is the size of the HDLC frame used. Each HDLC frame includes about three bytes of overhead in addition to the frame size, so that the effective throughput will be inherently reduced for small frame sizes. The test uses a kernel call for each HDLC frame. Thus the choice of frame size affects both the probability that the interrupts will occur while the kernel is executing and the amount of kernel processing required for each frame.

The test is conducted using two Gemini computers connected with an HDLC link. The application programs in one computer makes a series of kernel calls to transmit a sequence of frames and the other makes a series of kernel calls to receive the sequence of calls. Figure 2 shows the results of the series of tests. For all the tests there were no communications errors, demonstrating that the kernel was able to support the real-time response at all the transmission rates tested -- up to 64 kilobits per second. The amount of application and kernel processing per frame was constant for all the data points. For purposes of this test, unrealistically small

frame sizes are included to illustrate that even when the throughput is being limited by this processing, the character-by-character real-time response is still maintained.

SUMMARY AND CONCLUSIONS

The design of the GEMSOS security kernel for the Gemini commercial product has been influenced first by the Class B3 security requirements and second by the objective of high performance. We have described the major security design choices and believe that these provide an implementation that is particularly attractive for application in embedded systems. We have also reported the results of preliminary throughput and real-time performance measurements. These show effective real-time capability and nearly linear increase in throughput as the number of processors is increased up through seven processors. Although these are on an early version of the kernel that will be improved, we believe that these results already demonstrate that a secure system can also have high performance.

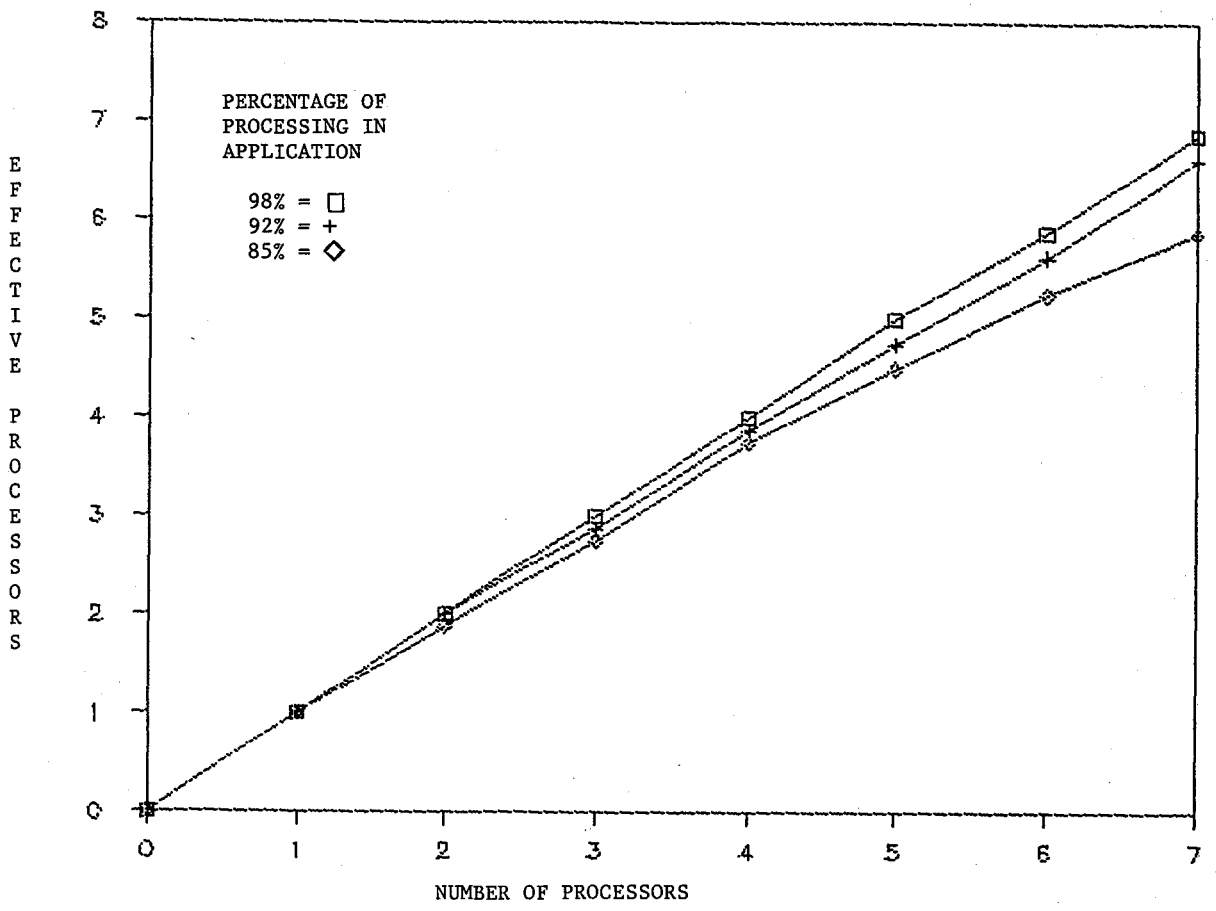


Figure 1. Gemini Multiprocessor Enhancement

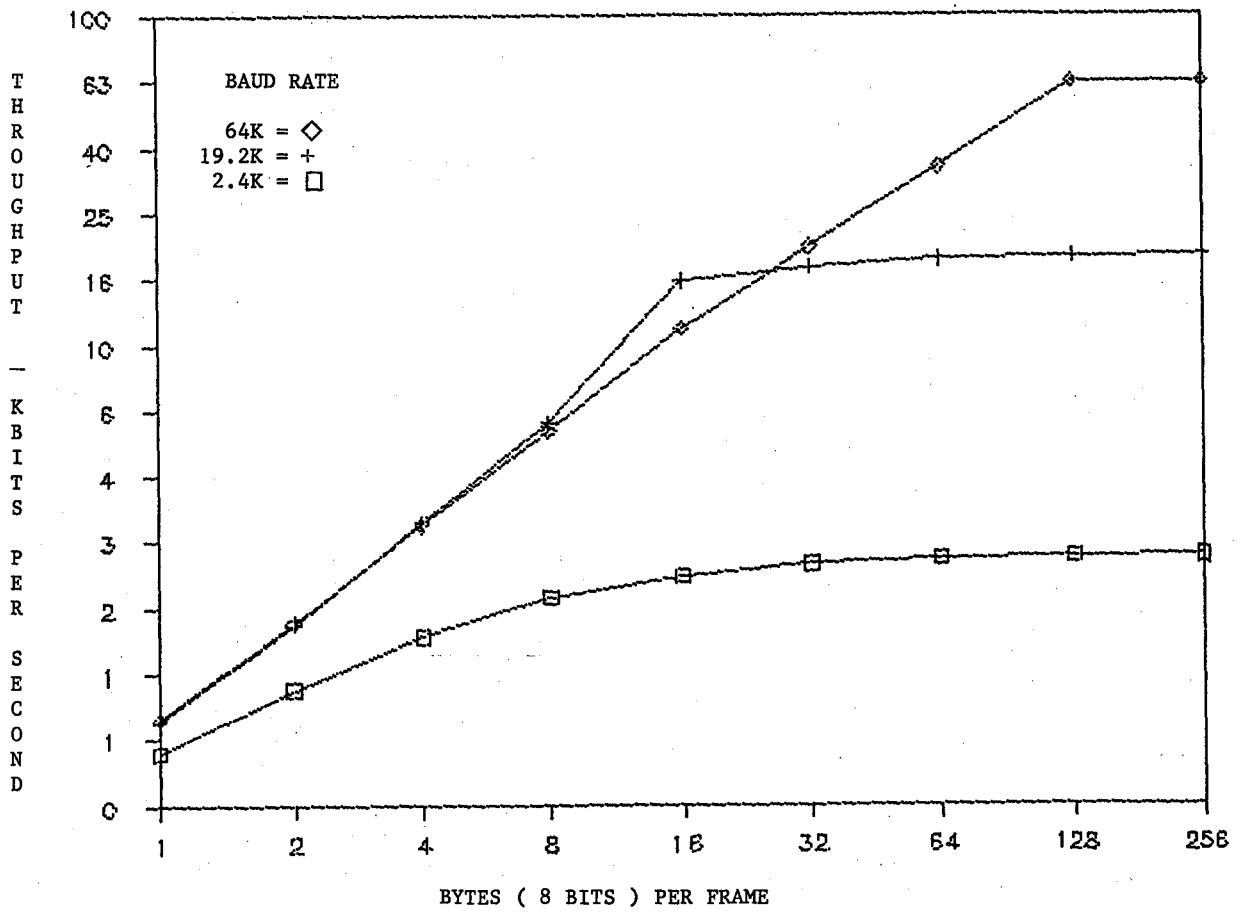


Figure 2. Gemini Real-Time Processing

REFERENCES

1. DoD Trusted Computer System Evaluation Criteria, CSC-STD-001-83, 15 August 1983, DoD Computer Security Center, Ft. Meade, Md.

2. Schell, R. R., and Tao, T. F., Microcomputer-Based Trusted Systems for Communication and Workstation Applications, Proceedings of the 7th DoD/NBS Computer Security Initiative Conference, NBS, Gaithersburg, MD, 24-26 September 1984, pp. 277-290.

3. S.R. Ames, M. Gasser, and R.R. Schell, "An Introduction to the Principles of Security Kernel Design and Implementation," Computer, Vol. 16, No. 7, July 1983, pp. 14-22.

4. D.E. Bell and L.J. LaPadula, "Computer Security Model: Unified Exposition and Multics Interpretation," Tech. report ESD-TR-75-306, AD A023588, The Mitre Corporation, Bedford, Mass., June 1975.

5. M. Schaefer and R.R. Schell, "Towards an Understanding of Extensible Architectures for Evaluated Trusted Computer System Products," Proceedings of the 1984 Symposium on Security and Privacy, April 1984, pp. 42-49.

6. K.J. Biba, "Integrity Considerations for Secure Computer Systems," Tech. Report ESD-TR-76-372, The Mitre Corporation, April 1977.

7. M.D. Schroeder and J.H. Saltzer, "A Hardware Architecture for Implementing Protection Rings," Communications of the ACM Vol. 15, No. 3, March 1972, pp. 115-124.

8. L.J. Shirley and R.R. Schell, "Mechanism Sufficiency Validation by Assignment," Proc. 1981 Symp. Security and Privacy, IEEE Cat. No. 81CH1629-5, April 1981.

9. D.P. Reed, and R.K. Kanodia, "Synchronization with Eventcounts and Sequencers," Communications of the ACM, Vol. 22, No. 2, February 1979, pp. 115-124.

SECURE SYSTEM DEVELOPMENT AT DIGITAL EQUIPMENT: TARGETTING THE NEEDS OF A COMMERCIAL AND GOVERNMENT CUSTOMER BASE

Steven B. Lipner
Secure Systems Engineering
Digital Equipment Corporation
Littleton, Massachusetts

INTRODUCTION

This paper is a brief summary of Digital's perspective on the problems of security in computer systems and networks, and on strategies for achieving improved security in commercial products. The paper begins with an overview of the security problems faced by Digital and its customers. It then presents an overview of development directions and options in the areas of operating system security and network security. Finally, it presents some thoughts on the problems and opportunities that result from enhancing the security of a family of commercial products.

SECURITY THREATS AND REQUIREMENTS

In thinking about the security requirements of our government and commercial customers, we have identified a rough characterization of the threats to system security and the associated range of countermeasures. We have come to think of the security threats to computer systems in terms of:

- o User irresponsibility;
- o Probing; and
- o Penetration.

User irresponsibility refers to the class of incident that occurs when an authorized user of a computer system takes some action that, while precisely within his or her area of authorized activity, constitutes disloyal or criminal conduct. For example, a bank teller might falsify the balance of an accomplice's account, or an intelligence officer might steal a copy of a report that he or she was authorized to receive. In each case, the user is abusing the trust that the computer system's owner has vested in him or her. Cases of user irresponsibility have received a great amount of publicity in recent years. But to the extent that user irresponsibility represents an abuse of trust by an authorized person, it is almost independent of the computer or communications system involved.

Probing is a term that we have "invented" to refer to cases where an individual takes advantage of a computer system that is poorly managed or whose security features do not allow it adequately to protect its resources. A system that is operated with unchanged "default" or distribution passwords is vulnerable to probing. So is one that only allows its users to share files with other users by making them accessible to all users on the system. Similarly, an uncontrolled broadcast local-area network is vulnerable to probing. In each case, one can take actions that are completely legitimate within the context of the computer system, but that have consequences that are unanticipated and unfortunate for the system's owners or managers. Most of the widely publicized incidents of "hacking" directed at computer systems appear to have fallen into the category of probing.

Finally, penetration involves completely circumventing or bypassing a system's nominal controls to achieve some unauthorized objective. One might, for example, write a program that exploits a flaw in an operating system's parameter checking to gain control of a computer system in supervisor or kernel mode. Or one might break into the wire closet in an office building to install a wiretap on a telephone line. While probing involves gaining access to information that is readily available, penetration typically involves the application of a relatively high level of effort and malice. Particularly in the area of computer security, the incidents of penetration that we have seen are in the nature of security test results, rather than actual malicious incidents.

If we consider the needs of the users of computer systems in the context of the three classes of abuse outlined above, we can draw the following conclusions:

- o Almost everyone who uses a computer or communication system to process valuable information must worry about user irresponsibility.
- o Users who operate their computer systems in open, shared, or exposed environments must worry about probing.
- o To date, the problem of penetration has been of concern primarily to users who process national security information in relatively open or exposed environments.

The classes of threat or abuse identified above seem to map reasonably well to the evaluation divisions and classes defined by the DoD Trusted Computer System Evaluation Criteria[1]. Since user irresponsibility is a problem of abuse of trust by an authorized individual, the need to counter user irresponsibility imposes

little burden on the trusted computer system. One installs one's controls at the application level, perhaps supplementing them with a Trusted Computing Base (TCB) of Class C1 or C2. To resist probing, in contrast, one must have a better set of security features in the TCB and they must be effectively used. The mapping to the criteria is perhaps to classes from C2 to B2. Finally, a very robust system indeed, in the B2 to A1 range, is required to address a threat of penetration. One might envision a similar mapping in the area of computer network or communications security, but in the absence of criteria, it is harder to identify a definitive structure.

SECURE SYSTEM DEVELOPMENT DIRECTIONS

The characterization of threats to system security is one major factor in determining the direction of Digital's efforts in secure system development. Given that many of the computer systems that Digital has supplied are used in environments where their features and interfaces are relatively exposed to manipulation by inside or outside users, it is appropriate for those systems to be able to withstand threats of probing. Thus it is appropriate for the preponderance of Digital's computer systems to achieve the high "C" or low "B" ranges of the DoD criteria over time.

The second major factor in determining plans and strategies for improved system security is the fact that Digital, as a commercial manufacturer of computer systems, has a large number of customers and Original Equipment Manufacturers (OEMs) who have developed software to run on existing operating systems and networks. Therefore, improved security must be achieved without sacrificing that software investment. The Digital strategy is to develop compatible security enhancements to its computer system products, rather than develop a new, incompatible line of "secure systems".

The following paragraphs give a few specific comments on Digital's development efforts in the areas of operating system and network security.

Operating System Security Enhancement

Digital began research efforts in computer security in 1979 with a project to develop prototype security enhancements for the VAX/VMS operating system. These enhancements were intended to meet threats in the probing category, and were aimed at meeting both commercial and national security requirements. In addition to providing a broad set of security enhancements, the project also had goals to maintain complete upwards compatibility with the then current version of VAX/VMS, to not require a major reimplementation of the operating system, and to have minimal adverse performance impact. The purpose of the research prototype was to evaluate the technical feasibility of these goals and to permit experimentation with various human interfaces to the security features.

The prototype explored enhancements to VAX/VMS in the areas of discretionary access controls, mandatory access controls, authentication, auditing, and system integrity. In each area, it was found to be possible to enhance the security of VAX/VMS without undue impact on compatibility, performance or human interface. The prototype effort was completed in early 1981, and the results provided to the VAX/VMS Development Group.

Expressions of interest by both government and commercial customers clearly justified an effort to improve the security of VAX/VMS. The research prototype pointed the way in terms of feature definitions, and a significant effort was undertaken as part of the development of VAX/VMS Version 4.0. Version 4 was released to customers in late 1984 and includes major enhancements in the following areas:

- Login and password management - a number of controls have been introduced to encourage users and system managers in good password habits, and to make it more difficult to guess a password and log in to a VAX/VMS system without authorization.
- Discretionary access control - an access control list mechanism has been introduced, supplementing the owner/group/world system, and allowing users to grant or deny file access to the granularity of an individual user.
- Auditing - a selective auditing facility allows recording of security-sensitve login and file access events.
- Integrity improvements - a number of changes have been made to reduce the possibility that a user will violate the system's controls and gain access to information without authorization.
- Documentation - the Guide to VAX/VMS System Security[2] consolidates information about the secure use and management of a VAX/VMS Version 4.0 system in a single handbook.

VAX/VMS Version 4.0 has been submitted to the DoD Computer Security Center as a candidate for evaluation at Class C2.

The prototype security enhancements to VAX/VMS included mandatory controls on major storage objects. As part of the development of VAX/VMS Version 4.0, preliminary work was completed toward the incorporation of labelled protection on files, directories, and certain other objects, and toward introducing the notion of user security "clearance." While these features are not yet supported parts of VAX/VMS, the development completed to date clearly indicates the

feasibility of a future version of VAX/VMS reaching Division B in the DoD Criteria.

Security Kernel Development

As the discussion above suggests, the demand for systems that can resist penetration has been relatively limited and specialized to date. Nonetheless, there does appear to be a growing set of users, especially in the national security community, for systems in the higher classes of the DoD Criteria. Accordingly, Digital has been exploring the possibility of developing a security kernel - a Class A1 system - for the VAX architecture.

It is a given that a security kernel for the VAX could not require the development of an entirely new set of layered products (compilers, data base systems, and the like), for the cost of such development would substantially exceed that of the kernel itself. Rather, the kernel would have to be compatible with the existing VAX/VMS layered products. In addition, because many Digital customers now use VAX/VMS to process sensitive information and have substantial investments in application programs and data bases, a kernel would have to support the continued use of those programs and data with no more change than is introduced in the transition from one major release of VAX/VMS to the next.

The remaining concerns pertaining to a security kernel for VAX are performance and verification. While users who needed a penetration-resistant system would probably be willing to pay a performance penalty, it is clear that they would not be willing to pay an unlimited price. A kernel for VAX should operate at a moderate performance penalty compared to VAX/VMS running on the same hardware. Finally, formal verification of any secure system poses a fearsome challenge - especially to an organization whose experience runs to operating system development rather than verification research.

Digital initiated advanced development of a prototype security kernel for the VAX architecture in mid-1982. The prototype embodies a strictly layered internal organization, and is tailored to provide an efficient environment for the execution of VAX/VMS application programs. It is implemented in the PL/I and Pascal programming languages, with machine-dependent and performance-critical components in assembler language.

The initial phase of the VAX kernel prototype development was completed in mid-1984 when the system achieved the capability of running VAX/VMS layered products and application programs without modification. Since that time, experiments with the prototype have centered on continued assessment of the system's user interface and security characteristics. Digital has also initiated a "developmental evaluation" of the prototype kernel with the DoD Computer Security Center.

Network Security Developments

The evolution of the Digital Computing Environment is taking us to the day when it will be unusual for Digital to sell a computer system without also selling a network connection. While there are not yet formal evaluation criteria for the security of computer networks, it seems intuitively clear that the classes of threats outlined above have their analogues in the network environment. It is also clear that, in most environments, encryption is required to protect information in a network.

Digital has developed a prototype end-to-end encryption capability for DECnet, based on the National Bureau of Standards Data Encryption Standard (DES). End-to-end encryption was chosen because of its flexibility with regard to alternative transmission media and "untrusted" network switches, and because broadcast networks like the Ethernet cannot support link encryption. Digital has been participating in ANSI and ISO working groups and committees on network security standards. We have also been examining encryption concepts that might be especially suited to local area networks.

Although our work to date has employed the DES algorithm, we are very interested in NSA's new way of doing business. Thus we have joined the Commercial Comsec Endorsement Program (CCEP) and are examining the implications of this program on possible future network security products and options.

OBSERVATIONS ON ENHANCING SYSTEM SECURITY

Digital has been working with the DoD Computer Security Center and its predecessor, the DoD Computer Security Initiative, since the late seventies. We have been involved with the CCEP for a much briefer period - less than a year at this writing. We have enjoyed our contacts with the DoD security evaluation teams, and we believe that our secure system development efforts have benefited from them.

Some of the factors that have influenced the character of our secure system development and evaluation activities may be of interest, both to other companies that are planning to develop secure systems, and to government agencies and contractors involved with the secure system development and evaluation processes. Among these are:

- The compatibility constraints on our commercial products, and on any systems that must interoperate with them are fairly high. These constraints can influence the cost, schedule and feasibility of developing a secure system.

- The lead times for the development of our products are relatively long. It is possible for government security requirements to change faster than we can build products, with the result that no product that we deliver complies with the requirements current at the time of its completion. Such a situation is neither in the government's interest nor ours.
- The product development process is a conservative one, relying on early and accurate estimates of development cost and schedule. The introduction of technology at the "research" or "advanced development" level of maturity is usually inconsistent with the development of a commercial product. Historically, formal security specification and verification have been at this level of maturity.
- A significant fraction of the market for our products is an international one. If building security into our products makes them subject to export controls that would not otherwise apply, that is an argument against the security enhancements. This is a tradeoff of which both we and the government must be aware as we define security requirements, export control policies, and product content.
- As a commercial manufacturer, we deal with classified information only in relatively contained areas. A secure system development effort that required us to handle such information in many of our design, manufacturing and support groups would have a major impact on our way of doing business, and be impractical.

Digital is expending significant resources to improve the security of its products. Our approach is an evolutionary one that is aimed at providing a very broad base of products that are resistant to probing, combined with selected high-security systems for use in environments where penetration is a significant concern. VAX/VMS Version 4.0 is the first product of this strategy. Over time, additional products will provide further demonstrations of our commitment to security.

REFERENCES

1. Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83, Department of Defense Computer Security Center, Fort George G. Meade, MD 20755, August 1983
2. Guide to VAX/VMS System Security, AA-Y510A-TE, Digital Equipment Corp., Maynard, MA 01754, September 1984

CAVEAT

This paper presents the opinions of its author, which is not necessarily that of Digital Equipment Corporation. Opinions expressed in this paper must not be construed to imply any product commitment on the part of Digital Equipment Corporation.

DIAL-UP SECURITY UPDATE

Eugene F. Troy, CDP
National Bureau of Standards
Institute for Computer Sciences and Technology
Building 225, Room B-266
Gaithersburg, MD 20899
(301)921-3485

INTRODUCTION

There have been many recent stories in the news media about the widespread ability of computer enthusiasts to get into other people's computer systems. The most common access path for these so-called "hackers" is via the common dial-up telephone and the communications ports which are connected to almost every business computer system.

How do we protect against this threat? This presentation describes a number of ways that more communications protection can be achieved for business computers. A wide variety of hardware devices are on the market today which can do a creditable job of protecting dial-up lines entering a computer. These devices perform the communications protection function in several different ways, which can be confusing to the potential purchaser. There is also a significant variety in the ways that the devices interact with users to perform their security functions. There are some traps for the unwary purchaser, also. Many of the devices tend to be inefficient or require the user to do additional steps that may not be acceptable. The prices vary considerably. Other features, particularly the level of protective strength, vary substantially among the devices.

This presentation will help the system manager to make an informed decision regarding whether or not additional security is needed for the computer system's dial-up lines. It will also help the manager determine what kind of mechanism is most suitable to provide the necessary level of protection. The six different hardware approaches that are presently available to perform dial-up security will be explored in detail.

ADEQUATE CONTROLS FOR DIAL-UP COMPUTER ACCESS

There are certain minimum controls which should be in place in a computer system in order to provide a good level of protection from intruders via dial-up communications. The advent of the hackers raised into public consciousness the potential vulnerability from this source, although the weaknesses have been there all along.

Dial-Up Versus Direct-Connect Access

Any user's terminal or printer is connected to a computer by means of some form of communications. The very nature of dial-up communications implies that the user may be anywhere in the world that the common-user telephone network reaches. It also implies that the computer must assume the job of screening incoming calls, because anyone in the world who comes into possession of the computer's dial-up port telephone number may attempt to gain access.

Direct Protection of Communications Circuits

Using hardware mechanisms to protect the computer's dial-up ports and its external communications is a fairly new idea for almost everyone who does not work with military or government secrets. If the communications circuits are directly protected, the organization is less dependent upon the routine and often weak operating system access control mechanisms to shield the system from intruders. As the sensitivity, criticality, and need for accuracy of the information in a system with dial-up capability increases, this special form of protection becomes more important, and can be performed by means of various hardware techniques.

Special Measures to Protect Dial-up Ports

Three security measures are extremely valuable in protecting a computer from the threat of intruders via the dial-up telephone system. Not all computer systems are presently able to provide these capabilities without modification to the operating system. If the three measures are not available, it is possible to provide overt protection by addition of special external devices which are discussed in the presentation. There is a fourth measure that may be used to protect the information being transmitted from disclosure or tampering.

Highly Effective Identification. The keystone of all access control is effective identification and authentication of users. This generally means the use of a well administered user name and password process. If these standard mechanisms are not available or are weak, a number of mechanisms can be used to provide this capability. Most external dial-up protection devices concentrate on this area.

Adequate Event Logging. The system's own journaling capability should be used to monitor communications events in order to identify user difficulties and intrusion attempts. If this is not possible, as is the case with many smaller systems, several devices can be fitted to perform the function in connection with access control.

Limiting "Brute Force" Attacks. Mechanisms that limit the effectiveness of "brute force" repetitive attacks will significantly reduce the likelihood of a successful attack from

an intruder. Brute force is the single most common approach that an unsophisticated attacker will use. Any mechanism which prohibits more than a very small number of log-on attempts per connection is very useful.

Protecting Information from Disclosure. In addition to access controls, it may be appropriate to protect the information being transmitted between terminal and computer from disclosure or tampering. It is very easy to intercept standard dial-up traffic by means of wire taps. It requires only a slightly more sophisticated intruder to modify and retransmit information that has been intercepted, for the purposes of fraud. Mechanisms that encrypt the information on the line can prevent this condition if it is viewed as a problem.

ONE-END" PORT PROTECTION: STRATEGIES AND FEATURES

If the internal software controls of the host computer are inadequate to protect it from penetration by dial-up intruders, a straightforward means of improving access control is to add an external device to the communications link which will perform an independent call-screening function. Typically, this type of device is totally independent of the computer itself.

Depending on the manufacturer's approach, the protection device may be placed on either end of the communications circuit. Most versions are installed at the host computer end, but some newer devices are designed to be connected to the user's terminal. Additionally, such a device may be designed to perform its function on the digital signal emanating from host or terminal, or it may be placed on the "analog side", between modem and telephone set. Some versions are even incorporated directly into a modem, as parts of a single unit. There are various reasons for these placements, depending upon system configuration and security needs.

The following discussion will separate these devices into two categories. First, the devices which may be placed on the host end of the circuit will be described. These devices are properly called "port protection devices", or PPDs. Second, a newer and more flexible type of device, called "security modems" will be discussed.

Direct Protection of Dial-up Lines -- Port Protection Devices

A port protection device (PPD) is any external device fitted to a communications port of a host computer, which is intended to provide the function of authorizing user access to the port itself, prior to and independent of the computer's own access control functions. It is specifically intended to help control terminal access when dial-up communications are used. Four primary features of PPDs are:

Password Tables. All PPDs require the user to enter a separate authenticator (essentially a password) in order to access the computer's dial-up ports. This set of password tables external to and independent of the computer's operating system is characteristic of PPDs and is available on all models. This feature is the primary protection given by PPDs. In effect, these devices do little more than establish password protection for the computer's ports.

Dial-back to Call Originator. Some users erroneously refer to PPDs as "call-back devices". This is incorrect, because not all PPDs either have or require that capability. The function of call-back or dial-back to call originator (the potential system user) is available on some models, but may have significant drawbacks in practice. Its purpose is to operate as a second level of user or port authentication beyond the standard PPD password table. In effect, this provides a second hurdle for the potential user to surmount before gaining system access. If call-back is used, the sequence of user connection is as follows: The user dials the computer access number and is connected to the PPD. The PPD requires the user to enter a PPD table password, and then hangs up the line. The PPD searches its table and, if the password is found, identifies the user's telephone number that matches the password. The PPD then initiates a return call to the user and, once connected, becomes passive in the circuit for normal operations. If the PPD does not find the password in its table, this error is logged in some way as a security violation.

Hiding the Port's Existence. A PPD may "camouflage" the computer's dial-up ports so that the identity or even existence of the computer is not evident to an unauthorized caller. This is commonly a side-effect of some password entry methods, but may be separately engineered. Some PPDs, which use "analog-side" placement in the circuit, respond with a synthesized voice when connected to the user. Other PPDs, which are placed on the digital side of the modem, may display special screens on the user's terminal upon connection that are either blank or ambiguous, and which require the user to know what to do next to gain access to the system.

Journalling of Security Events. It is desirable to log and examine security-related events which take place, especially in the dial-up communications circuits. This will provide the system administrator with the ability to make a measured response to any security threat. Many models of PPD provide some form of logging or other warning signal of dial-up attack. This varies all the way from display lights on the front panel of the device to the use of a personal computer's disk files to record all types of user connection information. Information that should be logged for a given system varies with the sophistication of system and local administrative requirements. For example, systems which use the call-back approach may need to record enough information to generate telephone usage bills to system users,

because the host incurs all telephone toll charges with this approach.

Typical Examples of Port Protection Devices

The National Bureau of Standards (NBS), which sponsored the development of this presentation, does not provide evaluations of products or services to the general public. Mention of products in this presentation in no way constitutes endorsement of them by NBS or the author. Products described below are typical members of a class, and are discussed for general information only.

PPDs range widely in terms of price, basic capability, and additional features. The potential purchaser must closely evaluate his/her security needs in terms of such devices, because any of them may be appropriate in specific circumstances. The easiest way to categorize them for purposes of discussion is in terms of the number of communications lines or ports a given PPD is designed to protect.

A Single-port PPD: Optimum Electronics DL-125. Many dial-up security applications require the protection of a single port in a very straightforward way. Examples would be personal computers with auto-answer modems set up for remote operation, or larger computers with only one or two dial-up ports. There are a large number of PPDs which provide basic password-table security on one port for a nominal cost. The Optimum Electronics DL-125 is one of these. It protects a single port by means of a 25-entry password table, with optional expansion to 100 entries. There is no call-back, camouflage, or logging capability. The unit is placed on the digital side of the modem, and uses the user's terminal keyboard for password entry. Because of this feature, passwords can be made up of the full 128-character ASCII set, which provides greater potential security than the numeric password used on PPDs placed on the analog side. The current price of the unit is $275. A separate modem is needed.

A Small Multi-port PPD: Backus Dial-Safe 3. It is recommended that for control purposes no more dial-up ports be assigned than necessary. If the computer uses less than 16 ports for this purpose, as is common with small minis or multi-user micros, there are a few PPDs which can handle all the lines at once. The Backus Dial-Safe 3 is designed to protect from one to three ports. Its user password table contains 65 entries, expandable to 150. This table also contains the call-back telephone number dialing sequence for each user. There is no port camouflage capability, but the Backus does provide a Centronics-compatible printer port for logging purposes. It uses the user terminal keyboard for password entry. This system costs $1295 total, which works out to $424 per port. No modems are provided.

A Large Multi-port Model: Digital Pathways Defender II. For larger systems, there are a few units which can handle upwards of 16 dial-up lines. These are rack-mounted units, which can be configured to meet a variety of needs. The PPD which has the largest capacity is the Digital Pathways Defender II. It has a standard configuration that will handle 48 ports, but it is incrementally expandable to 384. Its password and call-back tables accommodate 1,000 users, but may optionally contain up to 4,000. The Defender can use an IBM PC as its supervisory terminal and logger. It is very flexible in terms of the ways that users interact with it, permitting both terminal and telephone touch-pad entry of password information. Devices in this range typically have a number of other standard or optional security features, such as selectable tables which specify permitted user hours of operation. The Defender also has optional encryption and token-method user authentication (discussed in the next section). In the standard 48-port configuration, this unit costs $9,800, or $204 per port without modems.

A Small Modem with PPD Capability: Lockheed-Getex GTX-100.
A potentially important new trend in dial-up communications protection is the addition of protection features to standard items of equipment, such as modems, multiplexers, and port expanders. This can represent significantly improved security for lower cost than if the functions were purchased separately. The Lockheed-Getex GTX-100 is a standard Hayes-compatible asynchronous 300-1200 baud modem with port protection designed into it. This unit has table capacity to hold 16 user passwords and call-back sequences. Call-back is an inherent feature of the device, and is performed uniquely. The user dials up the device, which then prompts for the user's telephone number and hangs up. Then the GTX-100 searches its telephone number table for a matching password. If both telephone number and password are found, then it calls the user back and prompts for the password. If the password is given properly, the GTX-100 enters normal mode as a modem. If the password is not correct after a specified number of tries, the unit will flag the matching telephone number and refuse to connect with it until reset by the security administrator. This unit costs $800 to protect one port, which includes the cost of the modem.

Protecting Computers from the Terminal End -- Security Modems

Several new devices which represent a new approach to dial-up protection have recently entered the marketplace as part of the trend towards combining security functions with standard communications units. These devices are special-feature "security modems" for user terminals. The approach is similar to the Lockheed-Getex PPD described above, in that the devices incorporate security functions into an asynchronous modem.

Features that are characteristic of security modems include the following. They refuse to operate as normal modems for dial-out purposes until the user enters a specified password. Passwords are correlated in a secured table inside the modem with dial-out telephone number sequences necessary to

connect the user with specified host computers. The table also has the ability to the complete log-on sequence for transmission once connection is made. This simplifies the job of connection for users, because all they have to do is enter the appropriate password. The unit will then automatically dial the computer and make connection with a pre-selected user account. Users have no control over the connection information stored in the security modems. The security administrator can connect with these units remotely and change the information whenever desired.

Examples of Security Modems

Recent product announcements indicate that modem manufacturers have discovered the marketability of embedded security features. Several major vendors have added security into their modems, often at no apparent increase in cost.

An Asynchronous Security Modem: Racal-Vadic Maxwell 2400PA. This unit is typical of the new products being announced. It is a 300-1200-2400 baud modem which stores complete connection and log-on information for 15 accounts. The information is remotely entered by the security adminstrator. In addition, the unit has a secret serial number that can be checked as a terminal identifier by the host computer. It costs $995 for one port, including both security and modem functions.

A Single-port PPD and Security Modem: Cermetek Security Modem.
Perhaps even more indicative of the new trend is the Cermetek device. This is a standard Hayes-compatible 300-1200 baud asynchronous modem which can operate in three different modes: as a standard asynchronous modem, as a security modem on the user end of the dial-up circuit, and as a call-back PPD with modem on the host computer end. A low-cost device like this would work well with personal computers which may operate at times as dial-up terminals for host computers and at other times as auto-answer devices for polling or information entry by remote units. In the terminal "security modem" mode, the Cermetek can store up to 16 host computer telephone numbers and accounts. As a PPD, its table has room for 25 user passwords and telephone connection sequences. The Cermetek costs $695 for one port, including both security and modem functions.

"TWO-END" PROTECTION APPROACHES FOR ADDITIONAL DIAL-UP COMMUNICATIONS SECURITY

The "one-end" security devices discussed in the previous section were designed to improve dial-up access control by adding a password screen to the communications port. In higher-security systems, this level of control may still seem inadequate. More positive identification of the specific terminal or user may be needed. Or a measure of resistance to snooping or tampering with communications traffic may be needed. In these cases, the "two-end" approach is required. In this approach, there is a security device attached to or used with each user terminal plus a matching device or software attached to or used by the host computer.

Increased Security With Two-end Devices

When the "two-end" security device approach is used, the level of communications security can rise markedly and user convenience may improve, but there may be a substantial increase in cost plus other drawbacks. Further, there may simply be no risk basis for installing that degree of security in a given system. All these issues must be examined before any purchase decision is made.

Degree of Additional Security Afforded. Most of the techniques used for "two-end" security involve the use of highly complex algorithms uniquely associated with specific terminals or users. In some cases, these algorithms are coded onto integrated circuit chips which are encased in some form of "token", such as a plastic card. In others, the algorithms are embedded in the circuitry of a box that is connected directly to the communications link. The premise is that the hardware or software at the host computer end "knows" what algorithm or special token is associated with each user or terminal. The host can use this algorithm to perform a mathematical computation and then challenge the user or terminal device to do the same. If the results generated at the user terminal end match those generated by the host end, then the host has authenticated the identity of the communicating party with a high degree of certainty. This approach does not require that the user remember anything which may be written down or given to someone else. The units are designed specifically to prevent copying of the token or authenticator device.

Tradeoffs in Cost and Flexibility. In all cases, the "two-end" approach requires that each remote user or terminal possess a device that matches in some way with a device or software at the host. This substantially increases the cost to secure any given dial-up communications network. The costs for these systems vary widely according to level of security provided and other features. Costs can range as high as $6,000 per user-host link if sophisticated concealment of the traffic is needed in addition to access control. Most of the "token" authentication devices cost between $50 and $100 per user, not counting the equipment or software required at the host end.

Two-end security devices break out between those which provide user or terminal authentication (access control) and those which provide concealment safeguards against eavesdropping or tampering. The latter also inherently perform a strong access control function. The potential purchaser must determine whether the concealment function is necessary.

Devices in the "two-end" category are generally easier to use than the "one-end", primarily because no passwords must be remembered and connection delays can be lower. On the other hand, the approach is more complex. There are more items to break, become misplaced, install, and maintain.

User Authentication "Tokens"

Numerous new devices are based around the concept of a unique "token" for each system user. Each token has a special algorithm or some other unique and non-copyable identifier embedded in it. The host computer can challenge the user in some way that can only be answered correctly by means of the token.

There are two varieties of user authentication tokens. The first and simpler approach is hand-held, which requires no terminal attachments. The token may be in various forms. Some examples now on the market include a calculator with special circuitry, a "smart" plastic card which displays the authenticator continuously, and a photo-sensitive wand which is designed to read and interpret special terminal displays sent by the host. With this approach, the user must read the authentication information from a liquid crystal display (LCD) on the token and then enter it in the terminal when challenged. In some cases, the user must first read a challenge string on the terminal and enter it into the token via keys. The host reads the authentication information and compares it to the "right" answer it has generated before approving access.

The second authentication approach is simpler to use but may be more costly. It requires the user to place his/her token into a device connected to the terminal that can read the token and transmit information to the host. The token can be in the form of a plastic card or key with embedded microcircuitry, or in a less secure approach it can be a plastic card with a magnetic stripe.

Terminal Device Authentication Methods

Often, terminals which are placed in areas well protected by a physical security perimeter are used in a dial-up mode. Commonly, there is no need to make special identification of the users beyond normal log-on procedures, but it would be valuable to verify terminal location and identification. There are three basic approaches to positive identification of the user terminal by "two-end" techniques.

Many standard terminals or workstations already have provisions for internal terminal identifiers, also called "answer-back memory". These are either fixed and pre-assigned identifiers (hard-wired), or more commonly, memory locations in firmware that can be changed to the desired code sequence during terminal set-up. It is usually possible to secure the code once it is entered so that it cannot be read or copied by the user. The host system can send a standard ASCII code (ENQUIRE) to the terminal that will cause it to respond with the "answer-back memory" contents for authentication. Some commercial software telecommunications packages for personal computers (for example, Crosstalk XVI) have provisions to emulate this feature. Also, some modems (for example, the Racal-Vadic Maxwell described earlier as a security modem) have this feature.

A second approach to terminal identification involves the use of matching pairs of devices that are inserted in the communications circuit. One device is placed between the terminal and modem, and the other device is placed on the host computer's port. An example is the Microframe "DataLOCK and DataKEY" system. The DataLOCK is a four-port unit for the host end which is able to generate challenges to the small portable DataKEY units that connect to the terminals. Each DataKEY is uniquely encoded by the DataLOCK, and can be re-coded at any time. The DataKEYs also require physical unlocking by use of a standard brass key.
A newer version of the DataKEY unit, called the MagnaKEY, has a slot for magnetic striped cards, so that it can be used for user authentication as well. This represents the third approach to terminal authentication, in which a user's token can be inserted into the terminal authenticator unit. The "Codercard" is a similar approach, which requires each user to insert a thick plastic card with embedded identification circuitry into the terminal unit.

Encryption Devices

The process of encryption is simply "scrambling" information in a pre-determined way so that it is unintelligible to anyone who does not know how to "unscramble" it. This process has been used by governments for centuries to protect secrets while in transmission, but has been little used elsewhere. Increasingly sophisticated ways have been invented to do encryption, because attempts are always being made by intruders to "break the code". The newer encryption methods can only be done by computers or special microcircuitry.

There is a standard method that has been developed by the National Bureau of Standards for use within the Federal Government and elsewhere, called the Data Encryption Standard (more commonly referred to as DES). This method uses a highly complex algorithm that has been well known for nearly 10 years but has not been broken, although many mathematicians have attempted to do so. DES requires the entry of a 64-bit "key" sequence for encryption and decryption. Since each bit can be "on" or "off", this makes an extremely large number of keys possible, wherein lies the strength of DES. It is essentially impossible to use even computerized brute force techniques to discover the key used to encrypt a given message.

The use of encryption techniques for dial-up communications represents the highest form of security which can be applied to it.

Encryption has several attributes which cover most communications security needs. First, it secures the information passing over the communications link from disclosure to snoopers. This is the primary rationale for using encryption. Second, encryption effectively assures integrity of the message, so that tampering or inadvertent errors cannot take place. In modern, computer-based encryption processes, any modification of the encrypted message at any point will result in all subsequent portions of the message to be garbled, so the change will be noticed immediately by the recipient. Third, the uniqueness of the encryption key which must be shared by sender and receiver enforces an extremely high degree of user identification. If both sender and receiver share a single key, they must have exchanged it or been assigned it by a third party.

There is one common problem with communications encryption. If the key used by sender and receiver is the only real security, then the problem of exchanging keys so that both know which key to use at a particular time becomes paramount. Most encryption systems rely on the users to transfer keys manually in some way, which may or may not be secure. The intruder may have an opportunity to intercept the key while it is in transit. Encryption security quickly becomes a key management problem.

An Innovative Encryption Approach: Isolation Systems "Enigma". There are numerous encryption devices on the market. One interesting device represents a trend towards more practical applications of encryption because it handles the key management problem. This unit is the Isolation Systems "Enigma", which makes use of drop-in circuit boards for IBM PCs to create a secure dial-up network. Each board is pre-programmed by the system security administrator with a profile that specifies which of the other stations on the network each user may contact. Each Enigma board consists of encryption circuitry, a microprocessor with secured memory, and a standard modem with both auto-answer and auto-dial capabilities. The boards can communicate with each other in a secure way to exchange encryption keys to be used for a single communications session. If one user wants to connect with another to exchange sensitive information, the user calls up a special program and requests connection. The Enigma board then determines whether the user may make the connection. If so, the board places a telephone call to the other system's Enigma board, exchanges session keys encrypted in a higher-level encryption key the two boards share, and enters into the communications session with the session keys operative.

Message Authentication Methods

One "two-end" dial-up security approach has been designed specifically for electronic funds transfer (EFT), although it can be used in other applications. In EFT, it is important to verify that the contents of a message have not been changed, because these messages are in effect electronic checks which are subject to fraud or embezzlement. The banking industry, in conjunction with the National Bureau of Standards and the American National Standards Institute (ANSI), has come up with ANSI Standard X9.9 for Message Authentication in EFT. This standard uses DES encryption to encrypt selected fields in an EFT message to ensure that the message is not altered in transit. The encrypted information is brought together to form the "message authentication code" (MAC), which is then appended to the clear-text message to serve as a signature. The same process of partial encryption of significant portions of a pre-formatted message can be used in a number of business applications for verification purposes.

RECOMMENDED COURSES OF ACTION

A number of different alternatives for improving dial-up security via add-on devices have been presented. It is important to have a way of determining which, if any, of the devices have enough merit for the organization to warrant purchasing them. Each device provides enhanced dial-up security at some cost, in real dollars or in efficiency.

The problem of determinining dial-up security needs can be a very complex issue. Few persons outside of the military establishment are trained to make decisions about communications security. This section provides guidance on making the right dial-up security decision. The following set of evaluative questions should help focus the decision process and aid the system manager to settle upon a final course of action:

Does the Computer System Need Better Dial-up Security?

The first question to ask is: "How bad off are we now?" The following criteria are suggested to help determine whether the computer system even needs supplemental dial-up communications security devices:

Defining Security Requirements for Information Flowing on Dial-up Circuits. There are three measures which can be used to determine security requirements for collections of information or the systems which process them. The first is sensitivity to disclosure. There may be some negative impact that could occur if the information in the system were disclosed to unauthorized persons, such as dial-up intruders. This can be measured in terms of degree of impact. The second measure is criticality or availability. There often is impact to the organization if the information or processing system is not available within a specified period of time. The third security measurement factor is integrity or fragility to modification. If the information must be error-free to be useful or if it is the potential target of fraudulent modification, this factor is involved.

Characteristics of a Dial-up Circuit Needing Communications Security. Dial-up communications security devices can reduce organizational impact from all three security factors noted above, especially sensitivity and integrity. If the potential exposure via dial-up communications networks is high, that is, if intruders could gain access to the system to affect it or if they could tap or interfere with communications and thereby cause harm, then additional security protection is needed.

A dial-up circuit needing strong communications security is one that has one or more of the following characteristics: It passes data that must not be modified or disclosed, it supports processes with great time sensitivity, or it permits easy access to fragile data bases or files that must not be modified improperly.

If More Security Is Needed, Is One-end or Two-end Best?

If management determines that dial-up security devices are required in order to shore up internal host computer security capability, the next decision is about the general type of device. The following criteria are suggested to help decide whether the one-end (host or terminal port protection devices) or one of the two-end types of mechanism is best for meeting the computer system's security needs:

Sensitivity to Dsclosure and Integrity. If sensitivity of the information in the communications channel to disclosure or fraudulent modification is very high, then one of the two-end approaches which involves encryption should be used. If it is low to moderate, then a one-end approach which provides extra ability to screen out intruders via access control barriers may be appropriate.

User Resistance to Remembering More Passwords. If users are resistant to remembering extra passwords for access control, then one of the two-end approaches which involves automatic user or terminal authentication via insertion of a token may be approprate. Possession of the token is functionally identical to remembering a password.

User Resistance to Call-back Delays. If users are resistant to call-back delays, but higher levels of user identification and authentication are required, then a one-end device which does not have call-back may be appropriate. None of the two-end approaches use call-back, but some of them induce user connection delays by requiring the user to perform some process of receiving a challenge, processing it with the token, and then entering the result on the keyboard.

If PPD's Are Desired, What Features Are Needed?

If it is determined that additional security should be in the form of a low to moderate improvement in user access control (identification and authentication), then port protection devices (PPDs) or security modems may be needed. The following criteria are useful for selection and application of PPDs:

Access Security Versus Password Entry Methods. There are three basic methods of entering the password into a PPD, each with its own security or convenience considerations. Some units require the user to respond with voice to challenges, in such a way that a numeric password is formed. This is time-consuming and will not be appropriate for users who use direct-connect modems instead of telephone sets. Similar units require the user to enter a numeric password via the telephone keypad. The problems with this approach are that some terminals may not have keypads, and more importantly, the numeric password does not have enough possible variations to be highly secure. The third method of password entry is via the user's terminal keyboard. This approach permits far stronger passwords to be created, because any character of the password can be any one of the 128 characters in the ASCII character set. Even terminals with direct-connect modems can use this method.

Security Evaluation of Various Features. Two PPD features that are either standard or optional merit special discussion. An important feature that all units share is the procedure for changing security tables. Low-security PPDs permit this to be done either manually or via a connected terminal with no special external security controls. Higher security devices require a special password plus a physical key to enter the device into supervisory mode.

A much-vaunted feature of many PPDs that gives additional protection but has numerous drawbacks is call-back. Once almost synonymous with PPDs, call-back can serve as a second password hurdle, but in many systems the users may call in from any of a number of possible telephone numbers. Also, if the first PPD password procedure is strong, the second hurdle may not be needed unless management wants to strongly control the locations that dial-up users may call from. Major drawbacks include user connection delays, reversal of toll charges, and increased security table administration problems.

If Two-end Security Is Needed, What Approach Is Best?

If the straightforward user authentication features of the PPD do not meet the security requirements of the dial-up communications network, then one of the four two-end security device approaches may be appropriate.

Information Sensitivity. If the information transmitted on the dial-up network is so sensitive to disclosure that it should be protected against wiretaps, the best solution is some form of line encryption.

Information Fragility. If there is a strong need to make certain that the information cannot be tampered with during transmission, the specific solution is some form of message authentication (MAC) via selective encryption, although full-time line encryption achieves the same objective by hiding the information.

Terminal Location. If it is important to know that a specific terminal device is being used or that the communications come from a specific location, the best solution is use of existing terminal authentication capability (if available on presently installed user terminals) or a terminal authentication device. However, if all that is needed is a check on the originating location of the call, a PPD with call-back will also do the same job, possibly at less cost.

User Identification. If it is necessary to know with some certainty that a specific individual is accessing the system, one of the various user authentication devices will meet this need. Line encryption can also help, if the user is required to enter an encryption key in order to use the device.

What Are the Tradeoffs in Adding Dial-up Security Devices?

The prospective buyer of hardware for communications protection should carefully consider the potential negative impact of installing these devices in the organization. This impact can arise from several factors, discussed below. Communications protection devices typically cost several hundred dollars per line. In addition, there are signigicant potential problems in terms of user acceptance. Finally, these devices impose an increased burden on the organization in the form of additional administrative procedures.

User Convenience and Enhanced Security. Users may understandably resist the requirement for remembering additional passwords for PPDs or security modems. The requirement to carry around an authentication token, such as a card or wand, is perceived by the typical user as little different. Even more onerous than these methods is the set of administrative procedures associated with maintaining some manual forms of encryption key management. The typical system users may view any of these additional requirements imposed for the sake of security as unnecessarily burdensome unless they clearly perceive the necessity due to risk.

Similarly, any form of connection delays due to security will often not be taken kindly. These delays will be induced by the call-back procedures used by some PPDs. Other procedures, such as the manual entry of an identification string generated by a hand-held authenticator token such as a wand or card, will also generate connection delays of a minute or so. Granted, a minute may not seem like much, but it is strictly overhead and must be justified in the users' minds as a valid imposition on their ability to get their work done. There is a strong possibility that the computer and its associated security requirements (personified by the system security administrator) may be viewed as opponents and hindrances to the worker trying to get his or her job done. It is therefore important that additional security measures be fully justified by the level of risk to the system. It is equally important that users be properly educated on these risks and the clear need for additional security mechanisms.

System Management Effectiveness and Enhanced Security. When system security weaknesses are examined closely, the most common problems are seen to be administrative. In other words, more security potential is available in a system than the people who manage the system use effectively. This is especially true of the user account name and password scheme. The issue boils down to people problems. Imposing hardware protective devices typically will not cure that malady. Rather, this new approach may make it worse.

For example, consider what happens when an organization decides to install PPDs on the numerous dial-in lines attached to its primary computer. Immediately, the problems of managing these devices will surface. One obvious problem to be faced is that of managing an additional access control (password) system, separate from that used by the host computer. The procedures for assigning and changing passwords for PPDs should be rigorous, otherwise the real protection they can offer will be reduced. Usually, this means that more people will be needed to manage the system. This will be especially true if the organization takes this opportunity to separate out the communications security function from the computer security function.

As noted earlier, the additional security devices will cost substantial amounts of money. The bare minimum cost per port to install hardware protection seems to be about $200, and it can range into the thousands, depending upon approach and level of security desired. Along with this initial capital cost is the recurring cost of maintaining and repairing the devices. Other direct and indirect dollar costs imposed by these devices may include the following:

- User inefficiency (one minute per connection times many connections per year adds up quickly in terms of salary).
- Computer processing delay while token authentication takes place.
- Increased host computer telephone bill because dial-back procedures require session connections to originate at the host end.

All of the costs involved must be identified and estimated to determine the true cost of installing additional dial-up security protection. This final cost should then be compared to an estimate of present risk from damage due to dial-up intruders, to evaluate whether the new devices are warranted.

SUMMARY AND CONCLUSIONS

Both one and two-end dial-up security devices can provide a valuable increase in protection from intruders. In some cases, this protection can come at very significant cost, however.

The following conclusions may be drawn about this family of devices:

- The present dial-up security devices are a valid short-term strategy if the present security capability of the system is inadequate to meet the perceived threat from dial-up intruders. It is important to note that vendors are integrating these security functions directly into newer models of standard communications devices at little or no extra cost.

- These devices should supplement, not replace other security mechanisms. If present administrative procedures are weak, adding the devices may not be a valid strategy. The first requirement is to make full use of present operating system security capabilities.

- The devices can be used improperly or ineffectively. In terms of passwords, for example, they are subject to the same administrative weaknesses as operating system security features. It is also possible to install more security capability than needed.

The Bottom Line:

Dial-up communications protection devices should be considered if the system manager is unwilling to trust the computer's operating system security capability, when fully utilized, to keep dial-up intruders out of the system.

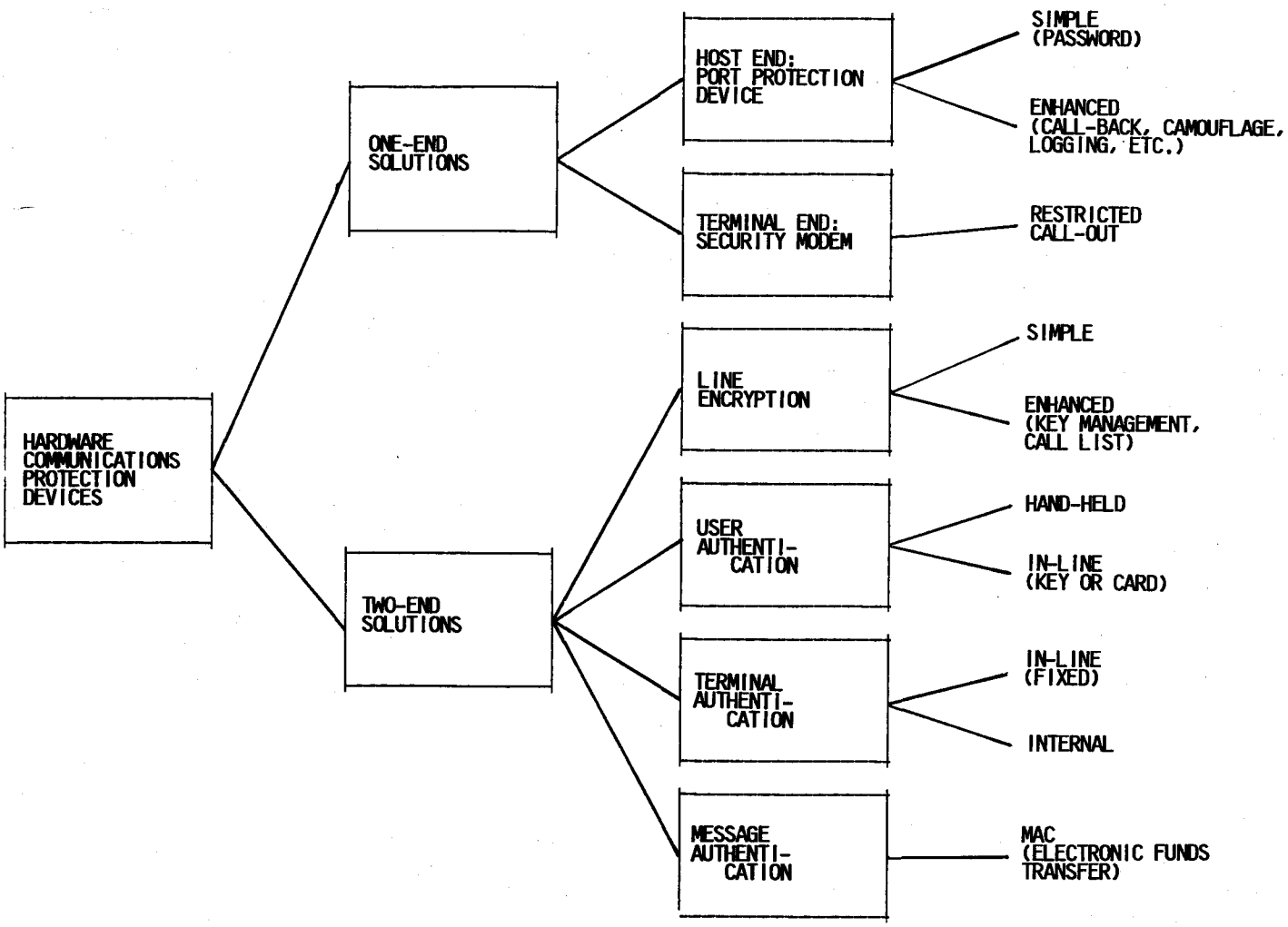


Figure 1. Hardware Communications Protection Device Alternatives

AN EMACS-BASED DOWNGRADER FOR THE SAT

John McHugh
P.O. Box 12194
Research Triangle Institute*
Research Triangle Park, North Carolina 27709

Abstract

This paper describes the design and verification of a downgrader to be supplied as a kernal extension for the Honeywell SAT. The downgrader implements a restrictive protocol based on traditional, paper document, downgrader procedures. specification and proofs techniques are described for both the security and functional aspects of the downgrader.

Introduction

The Honeywell SAT[1] (Secure Ada** Target) is a processor designed to meet or surpass the A1 level requirements of the Department of Defense (DoD) Trusted Computer System Evaluation Criteria[2] (TCSEC). It differs from previous systems which attempt to meet the criteria in a number of respects. The most important of these is the use of extensive, specialized hardware to provide the mechanisms for enforcing security policy while allowing the policy itself to remain in software to the greatest extent possible. The SAT is not intended to support a specific application, but rather to provide a vehicle for the construction of a diverse set of applications, military and otherwise, which have in common their need for a security policy and rigorous enforcement thereof. The downgrader discussed in this paper represents the first of a series of verified and trusted kernel extensions to be built for the SAT.

In this paper, we will briefly describe the philosophy behind techniques used to build a system on top of the basic SAT and show how this philosophy and guides construction of an extension such as the downgrader. With this background, we will describe the downgrader in terms of its functionality and operational interface. The process through which the downgrader is specified, designed, and verified is then described, followed by a discussion of the ways in which it fits into the overall SAT based system organization. Finally, it will be argued that incorporation of such verified trusted extensions in the SAT framework results in a system which can be shown to comply with an appropriate security policy in all of its operations.

Software and the SAT

The SAT is intended to support systems consisting of a number of user programs written in Ada. The hardware of the SAT provides the mechanisms for the enforcement of a security policy, but the policy is largely defined in software. The mechanisms provide support for a very general Meta Policy[3] of which the Bell and La Padula[4] policy is an instance. In providing a formal framework to support verified applications on the SAT, the Meta Policy is mapped to an abstract model level which is in turn mapped to an interpretation and ultimately to a formal top level specification (FTLS) which provides definitions of the actual machine instructions used to manipulate the security state of the SAT. Below the FTLS is a detailed design specification which defines the internal representations of the security state and the mechanization of the security relevant operations.

As noted in the SAT description, [1] complete verifications are performed at each of these levels and convincing arguments are made about the mappings between levels. These constitute the base proof for a SAT system. Addenda, which are proofs of software extensions to the SAT kernel, are combined with the base proof to produce a completely verified application.

The proof strategy of the SAT has been discussed in detail because the SAT design philosophy has been governed by provability considerations. These considerations continue to govern the construction of much of the user software to be run on the SAT to support a given application.

> There are two major classes of software in an SAT system: applications software, whose operation is mediated by the reference monitor, and kernel extensions, which extend the functionality of the base hardware to produce a reference monitor that meets the full and detailed DoD requirements.
>
> The first set of kernel extensions, and hence the first reference monitor to be built on the SAT hardware, will be those required for operation as an Ada Target. The initial et of users for SAT will then be system developers who wish to produce secure applications and more elaborate reference monitors in the Ada language. This choice of initial reference monitor characteristics was made in order to provide a secure, Ada-based capability to the development community as rapidly as possible.
>
> Consistent with this goal of timeliness, the first set of kernel extensions will be extremely simple. In general, kernel extension software is subdivided into three classes, based on the degree of trust and verification. (It should be recalled that "trust," in the sense used here, is the privilege to selectively violate the *-property, e.g., "write information down" in security level.) The three classes are:
>
> 1. Software that is neither trusted nor verified. Such software performs common resource management tasks, and its behavior is mediated in the same fashion as applications software. An example of this class

*This work was supported by Honeywell under U.S. Government Contract MDA904-84-C-6011.

**Ada is a registered trademark of the Department of Defense.

of software is the Ada Run-Time Support Library (RSL), which provides a virtual machine congenial to the semantics of the Ada language.

2. Software that is verified but not trusted. Certain kernel extensions must be verified to exhibit security-relevant properties, but these properties may not involve the benign violation of the *-property. Examples of this class of software are labelers, which must be shown to properly format exported labels, and login responders, which must be shown to properly consult a table of passwords before assigning a user name and a security level to a subject. Both modules perform functions that are security-critical yet do not involve information flows between security levels.
3. Software that is both trusted and verified. An example of such software are the tools that support the downgrading of information. Such tools must selectively violate the *-property and be verified to do so only in ways that are visible to and cleared by an authorized user. A secure downgrader, with an Emacs-like user interface, will be developed for SAT as a proof of principle that the basic SAT functionality simplifies the development and verification of such software.[1]

The downgrader discussed above is typical of kernel extensions which must be provided in order to produce a useful system for some application. It is patently obvious that such a facility violates the *- property of Bell and La Padula. It does not, however, violate the Meta Policy.[3]

In its broadest sense, the Meta Policy captures our intuitive motion of security. The policy can be simply stated as "information may not be disclosed to any individual without the necessary authorization."[3] It is important to note that, in this context, information is a semantic concept rather than a syntactic one and that the notion of authorization may involve the exercise of judgements based on semantics as well as syntactic constructs such as classification labels. This allows us to prove the security of the resulting system in a broader sense.

There are three sets of properties which must be included in a proof addenda for an extension such as the downgrader. The first of these is its functionality. The downgrader must be shown to enforce a protocol which is satisfactory for accomplishing the declassification of information when exercised by a trusted individual. It must then be shown that, as integrated into the SAT framework, the downgrader can only be invoked by such trusted individuals and, even then, only under suitably audited circumstances. Finally, it must be argued that this usage is consistent with our broader notion of security as represented by the Meta Policy.

The Downgrader

The downgrader is designed to force its user, a trusted individual, to follow a protocol which mimics the pencil and paper world. A primary objective is to ensure that the production of a draft document for downgrading takes place at human rather than electronic speeds. There are three phases in the operation of the downgrader. In the first phase, the user transfers information from a file representing a document at a high level of classification, H to a draft document, also classified at level H but ultimately intended for downgrading to a lower level L. This transfer is done a sentence at a time. During this transfer, limited changes can be made to the information being transferred. Following the transfer phase is a review phase in which both documents are reviewed in parallel with the context of each sentence displayed in both versions. During review, sentences in the draft may be accepted or rejected. Once review is complete, the draft document is copied down from classification H to classification L using the privileged TOPOP TWO (Trusted Write Override). The operations during each phase are discussed below in more detail.

Several aspects of the process are worth discussion. The most important is the fact that trust in the downgrader resides primarily in the human user who makes the judgments as to what information can be downgraded. The protocol serves to raise the level of assurance associated with the downgrading process by ensuring that the user has transferred information in relatively small chunks and under conditions requiring at least two inspections of each chunk. The process is also audited. Initiation of the downgrading process can create an audit record containing the identities of the user and the object being downgraded. Because the process operates on syntactic entities in the source object, the exact transformation of the source required to produce the downgraded object may be recorded for audit purposes. The actual downgrading of the draft may be made a System Security Officer (SSO) function.

The discretion implied above is deliberate. The downgrader design provides for capture of the transformations and for separating the transfer and review phases from the TWO. The SAT base provides the ability to restrict use of the downgrader and to preserve audit detail. The question of how much to restrict use of the downgrader is one of administrative policy and is ultimately a question of interpretation of the security Meta Policy.

The downgrader appears to its user as a two window screen editor which executes a set of commands which are a very restricted subset of those provided by EMACS.[5] In the transfer mode, the user may scroll through the source document at will, but may not change it. This assures a constant basis for the transformation history. The display is labeled with the classification of the object being downgraded as required by the TCSEC.[2] The user selects sentences to be transferred to the draft. In order to assure that the user is presented with a context containing the selected sentence, sentences are limited to approximately four lines. Limiting sentence length prevents simple subversions of the protocol such as converting a document to a single "sentence" prior to invoking the downgrader and downgrading it in a single operation. Restricting a sentence to a maximum of four lines ensures at least three lines of context on either side of the sen-

tence in each window (plus the security labeling required by the TCSEC[2] assuming a standard 24 line CRT display.

When a sentence is selected for transfer, it is highlighted or displayed in reverse video. The user then positions the cursor in the draft window which is labeled with the level of the source object and an annotation of the target level of the draft. The draft is displayed as individual sentences in cannonical form, separated by blank lines. New sentences can only be inserted at the beginning or end of the draft or between existing sentences. The user may scroll through the draft to determine an appropriate location to insert the transferred sentence. When the user issues the transfer command, the selected sentence is inserted in the draft window in cannonical form.

The cannonicalization process left justifies the sentence on the screen, converts all nonprinting characters to blanks and replaces all multiple blanks by single blanks. Line breaks are inserted as necessary. The purpose of the cannonical form is to prevent the covert passing of information to the draft document in the form of invisible data encodings such as sequences of nulls, backspace/blank pairs, etc. The user is left to deal with visible encodings although some of these could be included in the cannonicalization if desired.

At this point, the "whiteout" sub mode is entered. In this mode, the user can delete any word or sequence of words or can replace any word or sequence of words in the sentence with a cannonical marker. The words and phrases thus removed are saved for use during the review phase. Once the "whiteout" phase is finished, the sentence is considered as part of the draft. The user may delete sentences from the draft at any time or may re-enter the "whiteout' sub mode for any sentence in the draft at any time, but no other modifications are possible.

When the user has constructed the desired draft, the review phase is entered. At the beginning of this phase, the user reviews the list of deleted and replaced words and phrases. Unless the user explicitly eliminates a word or phrase from the list, it will be used as a review pattern, and all sentences in the draft containing the word or phrase will be singled out for special review in "whiteout" mode. During this part of the review, the user will be required to take an affirmative action to retain the word or phrase. These actions are subject to audit.

The general review causes the system to display the draft document in order while displaying the source document and context for each sentence in the draft. The user retains or rejects each sentence in the draft based on this review.

When the review is complete, the protocol has been satisfied and the actual downgrading of the draft may take place. The downgrading process may be suspended and resumed at will. Only the downgrader has access to the draft, ensuring its integrity. It is possible to revert to the transfer phase from the review phase, but the review must begin anew if this is done. It is also possible to abandon the downgrading as well.

Specification Issues

There are two issues to be addressed concerning with the specification and verification of the downgrader. The first of these deals with its security aspect. The downgrader must be trusted because it invokes a privileged SAT instruction, TWO when it creates the downgraded object from the draft. The power of the SAT's type and domain mechanisms allow us to define a lemma

```
AuthorizedDowngrader (In, Out)
```

which captures the security aspects of the proof. This lemma is formulated and proved as an extension to the SAT base proof and states that any downgraded object, `Out`, can only be produced by the downgrader, operating on the object, `In`, and executed on behalf of a user who is specifically authorized to have access to both `In` and the downgrader. The proof of this aspect of the downgrader's operation depends entirely on SAT properties and is independent of the functional aspects of the downgrader. Because of the use of the TWO operation to perform the actual downgrading operation, AuthorizedDowngrader is shown to be secure with respect to the Meta Policy.

Proofs of the addenda described above should be sufficient to show that the system with the downgrader incorporated still satisfies the security Meta Policy and is thus secure. One could, in fact, envision a much simpler downgrader which would merely do TWO on its input object and leave an audit trail. This too could be shown to satisfy the Meta Policy under the proper circumstances. Why then the elaborate protocol?

The answer to this question leads to the second issue which involves the specification and verification of the downgrader's functionality. Basically, although we trust the users who are allowed to downgrade, we do not expect them to be infallible. The trivial downgrader mentioned above provides no protection against such human errors as specifying the wrong object as input or overlooking a line or two during a review. The protocol does not completely obviate mistakes of this sort, but it makes them less likely and causes the downgrade process to be spread over a longer time, increasing the chances of catching an error before a policy violation occurs.

At the present time, the functional specification for the downgrader is under development. Like the SAT, the downgrader is being specified in Gypsy.[6] The functional specification presents a number of interesting issues. We want to show that the output of the downgrader bears a certain relationship to its input. This relationship say

```
ProperlyDowngraded (In, Out)
```

provides the desired traceability between the objects. This is not a sufficient condition for our purposes however, since

```
ProperlyDowngraded (In, In)
```

is necessarily true if only the document contents are considered. Thus, we need some way to link the procedural aspects of the downgrading process to the functional relationship between its input and output. This is further complicated by the interactive nature of the process and the use of a screen display to present the user with a window into the current state of the draft.

To support this notion, we start by considering the source as a sequence of characters. We map onto this, a syntactic index structure which creates a sequence of sentences of limited length as discussed above. This is a sequence of indices into the source identifying each sentence. The draft is a similar syntactic structure containing references to the source index and recording the modifications made in the downgrading process.

The user commands are obtained from a Gypsy buffer and are screened for illegal commands by a simple finite state recognizer. The display is described as an abstract type with a set of appropriate operations and "Hold" specifications. An

implementation probably requires trusted hardware for the display to ensure that the user and software "see" the same thing.

Given this model, the transfer phase is specified in terms of a finite state machine which as operations to select from the source, mark a source sentence for transfer, position the draft, move the marked sentence to the draft in whiteout mode, and exit whiteout mode.

It will be proven that every sentence in the draft is there as the result of applying this series of operations. Similar specifications and proofs will be supplied for other phases of the operation. The successful completion of a phase is recorded in the draft structure so that the definition of proper downgrading can be expanded to include the notion of the proper application of the sequence of operations required by the protocol.

The Downgrader and the SAT

Kernel extensions such as the downgrader play an important role in the construction of secure, useful systems on the SAT base. Such extensions may require verification either because they involve an element of trust as part of the TCSEC required reference monitor or because they enforce non-policy requirements, such as labeling imposed by the criteria. Verification may also be indicated when a high level of functional assurance is required due to nonsecurity aspects of the intended use of the system.

The SAT base proof provides support for all aspects of kernel extension verification for both trusted and untrusted processes by providing the formalisms necessary for the specification of processes calling on the SAT operations. In the case of the downgrader, a mixture of base addenda proofs and external, purely operational proofs will be performed. The net result will be to provide assurance that a SAT based system incorporating a downgrader enforcing an elaborate protocol to ensure that a downgrading procedure similar to that used in the pen and paper world conforms to an acceptable security Meta Policy.

It is important to note that satisfying the Meta Policy requires extending the notion of the trusted computing base to encompass trusted individuals, the authorized users in the case of the downgrader. Because the Meta Policy is expressed in terms of semantic rather than syntactic information, information flows such as those occurring during a downgrade do not, per se, violate the policy. Ultimately, it is the responsibility of the systems owner to define the level of assurance required (and the limitations on the trust to be placed in individuals) to show conformance with a given Meta Policy. The strength of the SAT approach is that it permits an owner to build on a proven base to extend the capabilities of the system in arbitrary directions while providing mechanisms to control these new capabilities and contain them within the TCB as necessary.

Conclusions

The Honeywell SAT and its use as a base for building secure systems has been discussed. A downgrader, functionally similar to the EMACS editor, has been described and its place in a secure system based on the SAT discussed. It has been demonstrated that this process, combining proofs of security and functionality, leads to a technique for implementing useful and trustworthy systems which meet or exceed the TCSEC criteria for A1 systems.

Acknowledgements: The notion to attempt design of the downgrader came from Earl Boebert of Honeywell. Marv Schaefer of the DoD Computer Security Center provided the idea of a Meta Policy and many of the details of the protocol in the guise of reviewing the initial proposal. Discussions with Earl Boebert, Marv Schaefer, Bill Young of the University of Texas and Scott Hansohn of Honeywell have been extremely useful in refining the ideas presented here. Any omissions and errors are, of course, my own.

References

1. W.E. Boebert, W.D. Young, R.Y. Kain, and S.A. Hansohn, "Secure Ada Target: Issues, System Design, and Verification," *Proceedings of the 1985 IEEE Symposium on Security and Privacy*, (1985).
2. "Trusted Computer Systems Evaluation Criteria," CSC-STD-001-83, Department of Defense (August 15, 1983).
3. W.D. Young, "Security in an Abstract Setting," Internal Note 186, Institute for Comptuing Science, University of Texas (July 1985).
4. D.E. Bell and L.J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretations," Tech. Rept. MTR-2997, MITRE Corporation, Bedford, MA (July 1975).
5. Richard E. Stallman, "EMACS Manual for Twenex Users," AI Memo 556, MIT Artificial Intelligence Laboratory (August 1980).
6. D.I. Good, R.M. Cohen, C.G. Hosch, L.W. Hunter, and D.F. Hare, "Report on the Language Gypsy, Version 2.0," Tech. Rept. ICSCA-CMP-10, Institute for Computing Science, University of Texas, Austin, Texas (September 1978).

MULTILEVEL APPLICATION DEVELOPMENT

Ronda R. Henning

Department of Defense Computer Security Evaluation Center
Fort George G. Meade, Maryland 20755-6000

DISCLAIMER

The opinions expressed in this presentation are solely those of the author and do not reflect the position of the Department of Defense Computer Security Evaluation Center or the Department of Defense.

INTRODUCTION

With the publication of the Trusted Computer Systems Evaluation Criteria (the Orange Book) in 1983, a renewed emphasis was placed on the development of trusted computer systems. Previous work done in the early 1970's on various secure computer system projects formed stepping stones that are the foundations of the current generations of secure computing systems. Considerable effort has been expended both by the Center and by various vendors to develop systems that meet the criteria for B2 class systems and beyond. One major emphasis in the infancy of the Criteria was that secure operating systems were feasible and existed within the scope of current technology. With the evaluation of the Honeywell SCOMP and the nearly complete evaluation of Honeywell's Multics operating system, it has been proven that multilevel operating systems can exist within the constraints of the real world. But now that the operating systems are available, what can be done to make them applicable to general purpose computer systems? What advantages do multilevel trusted systems have over single level trusted systems?

Multilevel systems are desireable in any environment that supports limited data sharing between users, complete segregation of groups of users on a single machine, or varying views of data for a single user or group of users depending upon the sensitivity level of the data and the user's operating level. Declining hardware costs make it less expensive to segregate users on separate machines with the sensitivity of the data the basis for segregation. However, for large applications requiring the incorporation of multiple levels (sensitivities) of data, it is less expensive to operate one trusted machine to support and segregate a large number of users than it is operate multiple untrusted machines that support one group of users each. Also, if limited communication is available between levels (ie, a channel of minimal bandwidth not exploitable by a potential penetrator) controlled data sharing and user communication is possible. Without physical media such as tape or hard copy output, such data sharing is not possible on separate machines. As local area networks are not presently able to segregate data by security level, they shall not be considered as a means of multilevel data sharing. Manual cut and paste techniques must be used to merge data from various sensitivities (levels), as opposed to having a trusted multilevel system merge needed data according to the users authorized access level.

In the case of the SCOMP and Multics, there are a series of routines that form the user interface to the security kernel. How the user tailors those interfaces to a specific application is his choice. However, the availability of common user software applications such as database management systems, bulletin boards, and other generic software applications (those which are used to build/tailor the system to a particular user environment) are virtually nonexistent in the true multilevel sense.

What are the implications of this lack of generic applications to the multilevel system user? In the present environment, the customer must write his own application development primitives as well as the application itself. The primitives are transparent to the user and other applications programmers in the final system design. They reside as internal routines within the finished application subroutines. For example, a routine find_node_in_tree would be an internal routine to a retrieve_token subroutine called by an application developer as a standard database subroutine/utility. If multilevel secure systems are to be made usable for general purpose computing, they must provide the tools for general purpose applications.

Realizing that there is a dearth of generic multilevel applications software, it is possible, within the framework of what is presently available, to develop software applications that function in a multilevel mode on B2 class and above operating systems. These present applications, as well as necessary and future areas for multilevel generic application packages and potential problem areas are discussed in this paper.

DEFINITIONS OF MULTILEVEL SECURITY

To understand the types of multilevel applications available at present and those needed for more general purpose multilevel applications, it helps to have a common frame of reference for multilevel security. The Criteria defines the phrase "multilevel secure" as a class of system that contains information of different sensitivities (classifications) that simultaneously permit access by users with different security clearances and needs-to-know but prevents users from obtaining access to information for which they lack authorization. Extending the definition to multilevel secure applications, a multilevel secure application can be defined as a type of application that contains information of various sensitivities that simultaneously allow access by users with different security clearances and needs-to-know but prevents users from gaining access to information for which they lack authorization. For the purposes of this discussion, multilevel applications are addressed as a separate issue as distinguished from multilevel operating systems. Generic multilevel applications development tools such as database management systems would, of course, require separate evaluation by the Computer Security Center to determine their compliance with the requirements of the applicable criteria or guidelines document.

Using this definition of a multilevel secure applications as a point of reference, there are two interpretations that represent opposite implementations. In the first interpretation, the underlying assumption is that the user functions at one and only one level and remains there for the

life of his account, hence his applications should function at only one level as well. This approach implies totally separate but identical copies of the same application at each level, with no shared files or data among the levels. This represents total data segregation. Multilevel secure applications are usually defined by vendors as totally segregated applications residing at various levels with no communication between levels whatsoever. For example, if a user worked a three different levels, to read his mail at each level he would be required to login and logout of each individual level. A user logged in at the Top Secret level could only read his Top Secret mail and reply to it. He could not read mail at the Unclassified or Secret level. The advantage to this approach is that there are a minimum number of possible covert channels and that multiple isolated users can coexist on a given computer system. The disadvantage is that there are probably an equal number of cases where controlled information sharing between the levels is highly desireable. This interpretation is most prevalent in the current generation of B2 and beyond systems and their application software, such as the Multics operating system and the majority of its application software.

A second interpretation of multilevel secure applications is one that allows controlled communication between levels with the user authorized to examine data up to and including his current operating level. As long as the communication is within the bounds of the Bell-LaPadua security model, it is permissible. For example, a user logged in at the Top Secret level could read his Secret mail; however, he could not reply to it and have the reply sent at the Secret level. This approach allows one data repository to exist for multiple levels, with data partitioned on a per user per level basis. That is, the user has his particular view of the data segregated for him dependent upon his current operating level. However, it does introduce a much higher number of potential covert channels. Multilevel users (and applications programmers) usually subscribe to this interpretation.

It should be noted that the two interpretations are not mutually exclusive. Multilevel applications adhering to the second (user) interpretation of the definition of multilevel security can be "layered" over a multilevel operating system following the first (vendor) interpretation of the definition. This is usually accomplished via an overseer process and multiple files residing at the various levels which correspond to the users authorized operating levels. The trusted overseer distinguishes among the various levels and routes data to the correct level file which matches the user's current operating level. Of course, this does imply that there is some degree of trust in the overseeing process, and does not account for potential covert channels that may be created during the routing of data to the correct level file. The fact remains, however, that it can be accomplished. Likewise, it would be possible to implement an application following interpretation one (vendor definition) on a operating system adhering to interpretation two (user definition). The data sharing mechanisms would have to be hidden from the user and disabled in such an application. Once again, there is a potential for the existence of covert channels. Because of these potential covert channels, it is best to implement applications consistent with the multilevel security policy/defintion used by the operating system. The applications and the operating system are separate entities, but the application must depend upon the operating sytem for its support functions.

TYPES OF MULTILEVEL APPLICATIONS

With the above interpretations of multilevel implementations in mind, it is worthwhile to discuss the types of applications suited to multilevel implementations. It should be readily evident that generic applications that are involved with data management functions are the most applicable to multilevel implementations because the great majority of applications are data management functions. Software such as database management systems, electronic mail, and bulletin board facilities come to mind almost immediately as candidates for controlled data sharing among levels. A database management system that would contain data from various levels and construct the user's current view to limit it to the data he is authorized to examine is a prime example of this type of generic application. From this example it is easy to see the applicability of a multilevel mode of operation to electronic mail and bulletin boards. A Multics Forum (bulletin board) could have transactions (entries) at various levels and categories consolidated into an entire view at the user-high level and decomposed as needed at the other levels.

Using generic multilevel applications such as database management as examples leads to a natural extension of multilevel applications to various user-written applications. These types of applications are principally in the areas of risk management and data segregation. For example, if the user wished to use highly sensitive corporate data in conjunction with widely available corporate data such as a list of employees by department, the application could avail itself of a multilevel database management system to combine the appropriate data at the level different from the general multipurpose user level into one view of the data at the alternate level. In this case, the database management system merges data from various separate files residing at different levels or creates a filtered view from the user high level, depending upon the implementation of the multilevel database management system. This approach would control the appropriate data by using the trusted database management system in conjunction with the trusted operating system.

Additionally, multilevel applications can function as a risk management tool. Those applications with highly sensitive data, for example sales projections or corporate revenue information, can function on the same computer system with the same generic trusted database management sytem performing the task of data isolation. This would minimize risk of inadvertent spillage of data to unauthorized personnel while eliminating any potential requirement for separate hardware or periods processing of this particular data. That is, a computer system would not have to be dedicated to any single level of data for its lifetime or a specified time period during a standard work day.

DEVELOPMENT IMPACTS

After identifying a particular application as a target for multilevel implementation, how does one approach the application software development process? It is imperative that sufficient time be allowed to do a throrough requirements analysis study of the project. Beyond the standard

requirements definition as used in structured design techniques, the requirements analysis should contain a precise definition of the entire system, with all user and external (other software not used specifically for this system) interfaces identified and detailed. Further, all multilevel data should be identified and the levels of application operation must be clearly defined. This does not preclude dynamic operation at the various levels, but does define the boundary levels for a given application. As an added precaution, any software that has been written for previous applications that may prove reusable (such as data format checking routines, etc.) in this one should be examined to ascertain its applicability and any nonmultilevel restrictions it may contain. Such recycled software written before a trusted multilevel generic application such as a database management system was in effect should be closely examined for unintentional covert channels and Trojan horses.

Armed with the completed requirements analysis, the developer should make an assessment of what can be expected in the way of multilevel support from the operating system. A system may be a B2 class or better operating system, but often system provided subroutines do not account for multilevel modes of operation because development is done at only one level, user low, and the software is never tested at other levels. The same can usually be said for vendor provided generic applications software. A valid assumption would be that the base system, including input/output routines, is about all one can guarantee will work with a reasonable degree of certainty. Additional features might not function in a multilevel manner depending upon the particulars of a given implementation, and such things are not always covered in the system documentation. Any assumptions about the user's particular level or default parameters are also invalid, as default values are usually based on the premise that the user works at a single level and remains there for the lifetime of his account. For these reasons, it is best for the developer to assume little in the way of support from system software and provide a application specific set of parameters for the application. The user's default parameters should be reserved and restored to him upon exit from a specific subsystem/application environment. The user should not have to worry about changing his default values for a particular multilevel application, nor should the user have to worry about the state his process is returned to upon completing use of a multilevel application package.

USER IMPACT

Once the application development staff has completed and tested a multilevel application, what can the user expect to encounter? Ideally the user will not have worked on the same operating system in a nonsecure mode. If this is the case, he will not have to relearn or unlearn features that do not function in a multilevel implementation and he will assume that the system's behavior was always the way it is in the multilevel mode of operation. However, if the user has worked on the identical operating system in a nonsecure mode, he will have some adjustments to make. Depending on how acquainted the user is with the nonsecure mode of the operating system, these adjustments could range from inconsequential to highly annoying. When given no alternative, however, the user will learn to function within the multilevel environment effectively in a relatively short time. The advantage to the user not having any prior knowledge of the operating system is that he does not have any way of knowing what features he cannot use as a result of multilevel operations. Users that have grown accustomed to features that do not work in a multilevel environment often have to overcome large amounts of frustration during the adaptation period.

ENHANCEMENTS TO MULTILEVEL SYSTEMS TO FACILITATE APPLICATION DEVELOPMENT

When one considers how few features of the operating system can be depended upon to function in a multilevel mode, it should be readily evident that the application developer on a B2 or beyond class of trusted system is working with tools comparable to stone knives and clubs in attempting to develop applications that are truly multilevel. Once a basic operating system has been evaluated, it is hoped that a vendor would also begin to make his generic applications software function in compliance with the same Criteria specifications. Even if the generic applications supported the vendor definition of multilevel security it would be a distinct improvement from the current situation, where generic applications can only be relied upon to function correctly at the user-low level.

What enhancements are needed to multilevel systems to make them more amenable to multilevel application support? If one examines most generic subsystems, such as database management systems and bulletin boards, they seem to have the same sort of operations in common, and perform these operations on various types of objects that tend to be smaller than a file. These operations characteristically are the same sorts of operations one does against a database management system, because the majority of applications are exercises in data management and manipulation. The general types of operations are the reading, modification, deletion, and creation of data to a previously defined data template and manipulate it according to a user specific set of criteria.

Since the majority of generic and user written applications perform the same varieties of operations, it would be logical to assert that the general interfaces to perform these manipulations should exist in one location and be accessible to any application. To that end, perhaps beyond the security kernel (or its equivalent) there should exist an applications kernel or applications interface to the security kernel. Such an interface could contain the primitives necessary for generic applications and customer written applications to function correctly in a multilevel mode. Alternatively, such an interface could be incorporated into the security kernel of the system. Placing the "applications kernel" within the security kernel would require its verification in the case of an A1 system, and would eliminate duplication of functionality. The argument for developing the applications kernel as a overlay to the security kernel proper is that those applications/users not requiring its features would not be forced to process their requests through this added layer of protection.

What precise functions would be included in this applications kernel that are not covered in the existing security kernel? The applications kernel would serve as a trusted filter to route data to its correct level in a multilevel system. It would serve as an arbitraitor and keeper of any necessary

lock tables for generic applications, and as a trusted data controller for user applications. With the applications kernel, it should be possible for the user application to specify the granularity level of object labeling. For example, a database management system could define the level of granularity as a tuple, while a bulletin board could define the level of granularity as a single transaction. This would be in addition to the labeling of the system file structure, which would be used by the arbitraitor to determine where to store the applications object given the file structure known to a particular application. In cases of conflict or the security mechanism being out of service, the arbitraitor holds onto the data and informs the database administrator or the systems security officer. Such an applications kernel could overlay the basic system security kernel and would be accessed only if the application required it. As a further precaution, manipulation of the applications kernel should reside strictly within a subroutine interface; it should not be accessible from command level. Embedding access to the applications kernel within application software keeps inadvertent penetration by casual users to a minimum, but retains availability of the features to the development staff.

Another potential problem with the applications kernel approach is the area of system performance. Mutilevel checking at the file level does not necessarily imply a performance penalty. In the case of the Multics operating system, the level checks are done regardless of whether or not the Access Isolation Mechanism is implemented in a multilevel mode or not. Whether or not level validation at the application object granularity would imply a performance penalty remains to be seen.

CONCLUSIONS

Multilevel application development with controlled data sharing between users at different levels is possible within the realm of current technology, although it is not widely implemented. If general purpose trusted systems are to become a reality, generic system application software such as database management systems must become available and function in a multilevel environment at more than one level. That is, one copy of the user application software and data must be accessible from various levels dependent upon the user's operating level. An applications kernel which would permit a finer level of granularity than the security kernel and enforce data segregation at this level is one method of implementing multilevel application software, both generic and customer specific.

It should be noted that this approach is not without its problems. In particular, there may be a severe complication with potential covert channels. The issues of data inference and data migration to the highest level an application can access do not go away either. However, the approach does provide the flexibility to facilitate multilevel application creation that is presently lacking in most trusted systems.

Only time and research will remove the present constraints on multilevel generic applications. Eventually, there will be trusted data management generic applications available. When such a time arrives, they will have to be evaluated in some fashion. The straightfoward solution would be to incorporate them into the operating system and evaluate them as one would evaluate any operating system under the Trusted Computer Systems Evaluation Criteria. An alternative would be to evaluate them as a separate product subject to a trusted application evaluation criteria. No doubt such questions will be addressed as the trusted criteria series evolves.

In conclusion, as secure operating systems evolve, no doubt secure applications software will also become available and grow beyond its infancy. Having a secure operating system is only the first step in a secure system. A series of generic secure applications is one step beyond a trusted operating system. The ideal trusted computer system environment will allow the customer/programmer working on a trusted system to develop his own trusted applications with no more effort or thought to security than would be necessary in the course of nonsecure application development.

BIBLIOGRAPHY

Air Force Studies Board, Committee on Multilevel Data Management Security, "Multilevel Data Management Security", National Academy Press, Washington, DC, 1983, pp. 1-60.

Arnold, Terry S., "The Practical Aspects of Multilevel Security", Proceedings of the Seventh DOD/NBS Computer Security Conference, 1984, pp. 30-37.

Chamberlain, Gray, and Traiger, "Views, Authorization, and Locking in a Relational Data Base System", Proceedings of the National Computing Conference, Volume 44, AFIPS Press, Montvale, NJ, May 1975.

Claybrook, Billy G., "Using Views in a Multilevel Secure Database Management System", Proceedings of the 1983 IEEE Security and Privacy Symposium, 1983, pp. 4-13.

Department of Defense, Computer Security Evaluation Center, Department of Defense Trusted Computer System Evaluation Criteria, 1983, CSC-STD-001-83

Fraim, Lester J., "Multilevel Security Today", Proceedings of the Fifth Seminar on the DOD Computer Security Initiative, May 1982, pp. 233-239.

Friedman, T.D., "The Authorization Problem in Shared Files", IBM Systems Journal, Vol. 9:4, 1970, pp. 258-280.

Rushby and Randall, "A Distributed Secure System", Proceedings of the 1983 IEEE Security and Privacy Symposium, 1983, pp. 127-135.

Schell, R.R., "A Security Kernel for a Multiprocessor Microcomputer", Computer, Vol. 16, No. 7, July 1983, pp. 47-53.

Schell, Ames, and Gasser, "An Introduction to the Principles of Security Kernel Design and Implementation," Computer, Vol. 16, No. 7, July 1983, pp. 14-22.

A PARTIAL SOLUTION TO THE DISCRETIONARY TROJAN HORSE PROBLEM

W.E. Boebert C.T.Ferguson

Honeywell Secure Computing Technology Center
Minneapolis MN

BACKGROUND

Discretionary access controls are attractive for applications in which the adminstrative overhead of a mandatory policy would be onerous. Such applications arise in situations where the responsiblity for maintaining the security of individual items of information is highly decentralized. The attractiveness of discretionary controls is counterbalanced by the fact that they are inherently vulnerable to Trojan Horse attacks. A detailed scenario for a Trojan Horse attack is given in Reference 1.

In this paper we show how the underlying mechanisms of the Secure Ada Target (SAT) machine we described in Reference 2 can be used to reduce the threat posed by discretionary Trojan Horse attacks. We call our approach a "partial solution" because it rests upon deterrence resulting from likelihood of exposure. In this regard it differs from the complete solution provided by enforcement of the simple security property and the *-property.

UNDERLYING MECHANISMS

Auditability

The SAT machine embodies the reference monitor model of computation at the hardware level, with clearly recognizable subjects and objects. A hardware register carries at all times a universal identifier (UID) of the user on whose behalf the current instruction is executing. It is not possible in the SAT machine to execute an "anonymous" instruction, and all operations performed on behalf of users are therefore auditable.

Ownership

Every SAT object has an owner, who is the only individual authorized to perform administrative actions such as deletion. Ownership may be passed by means of a "handshake" protocol, in which one individual relinquishes ownership and another accepts it. Thus a conceptual "ownership" token for a given object may be passed among the user community. The initial owner of an object is the user on whose behalf the object was created. Object creation and transfers of ownership are auditable events; it is therefore possible to create an audit trail which lists all owners an object had since its initial creation.

Type Enforcement

In Reference 3 we describe the mechanisms the SAT hardware provides for the enforcement of data typing. In this discussion it is sufficient to note that every object in an SAT machine is assigned a type and every subject has a domain as part of its definition. Part of the policy enforcement mechanism in SAT is a table by which accesses to individual types may be restricted to specified domains. The type enforcement

mechanism is used to encapsulate type managers into domains, and insure that no operations may be performed on a type save those implemented by its manager. Type enforcement is verified to the same level of assurance as mandatory security policy.

An important set of types are the subtypes of the general type "procedure." Creation of a procedure in the SAT system requires authorization and is performed by verified programs encapsulated in a specific domain. It is not possible for ordinary, unverified programs to convert data (such as the output of a compiler) into a procedure. Installation of a procedure is an auditable event. As well as providing a defense against the so-called "virus" variant of the Trojan Horse attack, this mechanism permits the construction of an audit trail in which the entire "pedigree" of a procedure may be reconstructed: initial creator and subsequent owner of the source file, installer, past owners and current owner of the procedure itself.

It is further possible in the SAT machine to encapsulate the interface to the hardware reference monitor in a domain. This domain may contain verified programs which audit or pass on the acceptability of requests to the reference monitor.

The Global Object Table

All object attributes are carried in a file call the Global Object Table (GOT), which is private to the hardware reference monitor. Typical attributes are the UID of the object, the UID of the procedure which created the object, the object's security level, access control list, and type, and the UID of the owner of the object. In the case of subtypes of the general type "procedure," the GOT entry contains the UID of the user on whose behalf the procedure was installed. The GOT is indexed by UID; given a UID, programs executing in privileged domains may request attributes of that object from the reference monitor.

USE OF THE UNDERLYING MECHANISMS

Approach

We shall now sketch the design of a type manager whose facilities increase the likelihood of discovery of a discretionary Trojan Horse attack. For purposes of example, we will describe the approach in terms of the construction of an audit trail. In more sensitive applications it is possible to have the type manager intercept, instead of merely record, actions which indicate that a Trojan Horse attack may be underway.

Our general approach is to encapsulate a dynamic linker in a domain, and interpose that domain between unverified procedures and the reference monitor. All procedures running on SAT must go through the reference monitor in order to obtain operand addresses; we now require that they go through the linker as well.

Dynamic Linking

Our approach uses the standard dynamic linking technique for systems which, like SAT, use subject-local addresses in instructions. In this technique, procedures are kept in "pure" form; one unmodified copy of the code is shared among all subjects in whose context instances of the procedure may be executing. References to global objects (i.e., those not "prelinked" as part of the procedure) are made indirectly through "links" which are collected into "link vectors." There is one link vector for each procedure instance. For example, consider the case in which instances of procedure P1 are executing in the context of subjects S1 and S2, and instances of procedure P2 are executing in the context of subjects S2 and S3. Then there will be four link vectors: P1.S1, P1.S2, P2.S2, and P2.S3.

There are two operations which are performed on link vectors: "link initialization" and "link snapping." A link must be snapped before it can be used as an indirect address, and it must be initialized before it is snapped.

Link initialization in SAT requires that the symbolic name of a global object be converted to a UID; this is done by reference to a global directory structure. Link snapping requires that the UID be converted to a subject-local address; this is done by the hardware reference monitor as described in Reference 2. Once a link is snapped it becomes a conventional indirect address. Link initialization occurs when the procedure instance is bound to a subject and link snapping occurs upon the first attempt to indirectly address through the link.

Operation of the Enhanced Linker

We now enhance the linker whose operations we described above in such a way that it can detect a large class of potential Trojan Horse attacks. The first enhancement involves a per-user table in which each user lists the names of other users he or she trusts not to mount Trojan Horse attacks. We will refer to such users as "fellow citizens" of the user in whose table they appear, and will call all other users "aliens." This data table is given a distinct type and is accesible only by the enhanced linker, all of whose code must be verified. Changes to the table require authentication of user identity and are audited.

Detecting potential Trojan Horse attacks now becomes a straightforward exercise in comparing UIDs from various sources. At the procedure. If so, the link proceeds unaudited; if not, an audit record is written.

PROPERTIES OF THE APPROACH

Subject Independence

It should be noted that the series of comparisons which are made do not involve any element of the subject in whose context the linking takes place. The relationship of interest is that which exists between between the owner of the data and the originator of the program: whether or not the programmer is known to, and trusted by, the owner. Thus the linking, and the tests, may take place with equal effectiveness when the program is invoked by some (possibly unwitting) third party.

Proof Requirements

Assurance in the enhanced linker requires proof that the linker cannot be bypassed and that it performs its functions correctly. The proof that it cannot be bypassed relies on lemmas previously proven about the SAT hardware, and in outline follows that of the labeller described in Reference 3. The proof that it works correctly involves straightforward application of program proof technology. The complex functions of extracting owner and originator UIDs, and comparing UID values for equal, are performed by the SAT hardware and are proven correct as part of the certification of that hardware. The properties which remain to be proven about the linker software itself are then simple in the extreme.

Performance

The approach described above extracts a performance penalty in that domain change overhead is incurred each time the linker is invoked; if it were not necessary to protect the enhanced linker and its tables from tampering, the linker could reside in the same domain as unverified code.

Vulnerabilities

We would like to conclude by reiterating that the above approach provides only a partial solution to the problem of discretionary Trojan Horse attacks. It is possible to devise scenarios in which attacks occur undetected. Such scenarios are, however, much more elaborate and (from the point of view of the attacker) much riskier than the simple offering of a subverted "utility" program which forms the basis of an attack on conventional systems.

REFERENCES

1. W.E. Boebert, R.Y. Kain, and W.D. Young, "Secure Computing: The Secure Ada Target Approach," Scientific Honeyweller, June 1985. Copies may be obtained from the authors at the Honeywell Secure Computing Technology Center, 2855 Anthony Lane S., St. Anthony MN 55418.

2. W.E. Boebert, R.Y. Kain, W.D. Young, and S.A. Hansohn, "Secure Ada Target: Issues, System Design, and Verification," Symposium on Security and Privacy, IEEE, 1985, 176-183.

3. W.E. Boebert and R.Y. Kain, "A Practical Alternative to Integrity Policies," these proceedings.

ACKNOWLEDGEMENTS

This effort has been supported by US Government Contracts MDA904-82-C-0444 and MDA904-84-C-6011.

A STATUS REPORT ON THE DEVELOPMENT OF NETWORK CRITERIA

by

Sheila Brand

DoD Computer Security Center

Ft. George G. Meade, Maryland 20755-6000

1.0 Introduction

The purpose of this paper is to describe the steps that have been taken by the DoD Computer Security Center (the Center) to develop guidance in the area of security of computer networks. The Center's Invitational Workshop on Network Security will be discussed along with how results of this meeting are being used to prepare draft Trusted Network Evaluation Criteria. The paper will close with a brief overview of the emerging network criteria and the major differences between this document and the DoD Trusted Computer System Evaluation Criteria. (4)

2.0 Background

In August of 1983, the Department of Defense Computer Security Center published the Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83. (4) (To be referred to as the Orange Book) That document provided a basis for the evaluation of the effectiveness of security controls built into automatic data processing system products. Though the Criteria defined in the Orange Book are application-independent, it was recognized early-on that specific security feature requirements in that document would have to be interpreted when applying them to applications and other special processing environments.

Soon after publication the question arose as to whether or not guidance in the Orange Book was sufficient for the evaluation of computer networks and network components. Debate has raged over this issue for the past two years with both points of view being strongly adhered to. In order to provide definitive guidance on network security it was necessary for the Center to address this issue head-on. The strategy was to examine security related issues involved in the evaluation of computer networks with the objective of publishing network guidance in one of two ways: (a) by developing a totally new set of criteria, or (b) by extending and/or revising the Orange Book so that it can be used unequivocably for the evaluation of computer networks.

In January of this year the Center's Division of Standards identified a series of issues whose resolution would bring the process one step closer to the development of network security criteria. It was recognized that for these issues to get a fair "airing," pooling of the Nation's scarce resources in this area would be necessary. It was therefore decided to organize a workshop, invite the Country's computer and network security experts to it, and use the Center's issues as the focal point for the workshop.

2.1 The Issues

The issues requiring resolution covered eight areas:

* Policy and Models
* Access Control
* Accountability
* Network Architecture
* Configuration Management and Testing
* Verification and Covert Channel Analysis
* Network Components
* Denial-of-Service

The complete list of issues are presented in the Proceedings of the workshop. (1)

To address the issues in advance of the workshop the Center invited a number of the Nation's network and computer security experts to write issue papers. Twenty-nine papers were written with an approximate distribution of three papers per issue area.

2.2 The Workshop

The DoD Computer Security Center Invitational Workshop on Network Security was held in New Orleans, Louisiana, 19-22 March 1985, with the stated objective of providing the Center with input necessary for the development of Trusted Network Evaluation Criteria. In addition to the issue paper authors approximately 50 more experts in network and computer security representing both the public and private sectors were invited to participate.

Each of the invitees was assigned to one of eight specific working groups organized around the eight issue areas and remained with that group throughout the Workshop. Each group was asked to read and discuss pertinent issue papers and provide criteria recommendations based on group findings. Each Working Group leader was asked to prepare a group report and provide that report to the Workshop organizer before leaving New Orleans.

3.0 Workshop Results

With a few exceptions the overwhelming consensus was that the Orange Book is alive and well and, yes, useful for the evaluation of the effectiveness of security controls in networks. However, extensions, interpretations and some additional criteria are necessary. The prevailing reasoning was that the distributed nature of a network allows it to be analyzed as a special case of a distributed system. As the Orange Book does not preclude, but also does not specifically include, distributed systems much interpretation is needed when applying Orange Book criteria.

Some of the conclusions reached by the working groups (and summarized by issue area) include the following:

3.1 Policy and Models

Security policies must be stated for: protection against compromise, for integrity, and against denial-of-service. Furthermore, these policies must be implemented at both the network level and at the component level. In the area of compromise the network mandatory policy should be enforced by a set of interconnection rules that provide the conditions under which two components can communicate. These rules are expressed in terms of the security level of the information being transmitted and the accreditation range of the sending and receiving network components. (7)

3.2 Access Controls

In this author's opinion one of the most important findings of the access control group was their recognition and emphasis on the need for standardization of labels within a network. They concluded that if a network is expected to make access control decisions based on the sensitivity level of the data for which transmission service is being requested, (i.e., mandatory access control decisions) then all subscribers of the net had better use the same representation for a specific sensitivity level. They went one step further and suggested that not only must internal representations be exactly the same but external label representations also. Their fall back position was that mutually communicating systems be required to maintain a mapping of each other system's labels to the other.

In addition to their recognition that mandatory access control can be implemented in a network, the access control group also suggests that some form of discretionary access control internal to the network is crucial for correct network functioning as well as for security. This is because it is closely tied to the problem of correctness of received identity. (8)

3.3 Accountability

The Accountability group recommends that the basic principle of individual user accountability must be supported by the network just as it is in hosts. They

recognize that to accomplish this task networks will require the cooperation of hosts to trace network activity back to individual users.

This group's report also discusses the nature of a network reference monitor. That portion of the network that is responsible for enforcing the network security policy is referred to as the Trusted Network Base (TNB). The TNB, which a network will have only one of, will include a network reference monitor. The TNB will be distributed over the network with parts of it residing in hosts which are attached to the network, other parts may reside in cryptographic devices, front ends, packet switches, etc. (3)

3.4 Network Architecture

This group's consensus was that the Orange Book is not adequate for evaluation of networks and that additional criteria are needed. The new Criteria must allow incremental evaluations which would allow an evaluator to examine the parts of a network that are used as building blocks as well as allow for evaluations of network in their entirety.

Some notion of formal decomposition of a network is necessary. This decomposition would require that when a system is partitioned into subsystems a security policy must be derived for each subsystem. The derived subsystem policy may not be the same as the containing system's security policy or adjacent subsystems policies. However, the security policies of all subsystems within the containing systems must be shown to completely satisfy the system security policy of the containing system. (10)

3.5 Configuration Management and Testing

The group recommended use of configuration management from the onset of network design if possible, but at the earliest stage possible. This should be instituted for networks of all evaluation classes from the lowest class on upwards. Support must be a global responsibility. That is, it is crucial that all security-relevant components of a network be integrated under configuration management so that a global authority can evaluate how proposed components changes would effect network security. (5)

3.6 Verifications and Covert Channel Analysis

As was the case with the network architecture group this group's findings emphasized decomposition. They viewed the network as a special case of a distributed system. From this perspective system-level requirements should be decomposed to yield component-level "constraints" (read: component security requirements).

The report emphasized the greater potential for exploitation of covert channels in a distributed system and recommends that all hosts connected to multi-level networks should be subjected to covert channel analysis, not just those at the B2 or above evaluation class. This group recommends that the definition for covert storage channel be expanded from that in the Orange Book. (9)

3.7 Network Components

This group believes that only with respect to or in the context of a specific network environment should components be evaluated with respect to system level security requirements. This will allow a network architect the latitude to meet system level security requirements in a varity of ways. A specific security requirement may be satisfied by one component, a homogeneous ensemble of components or by a heterogeneous collection from a variety of vendors. (6)

3.8 Denial-of-Service

Their major findings: (a) Integrity and authenticity of control are of utmost importance in coping with denial-of-servie problems. (b) The Orange Book is of little help in this area. (c) No generic denial-of-service conditions could be identified which were independent of mission objectives. (d) No increasingly comprehensive subsets of denial-of-service were identified but some categorizations in terms of detection, recovery, and resistance was done. (2)

Though the group was unable to recommend generic criteria they did provide mission oriented criteria starting with a general policy which is: "Denial-of-service requirements will be considered for all networks relative to the user mission being supported by that network. Each network provider, in cooperation with the user, will define what conditions constitute network denial-of-service."

4.0 Network Criteria

Following the workshop, the Center asked a number of people to assist in analyzing workshop results. The objective was to identify emerging trends and themes and to provide an assessment of the direct applicability of workshop products to the development of network criteria. As a result of this activity the basic outline for the criteria was laid out. The remainder of this paper will provide an overview of major features of the draft document, titled: Department of Defense Trusted Network Evaluation Criteria. (TNEC)

4.1 Definition

The draft TNEC uses the following definition to describe a network: a network is composed of a communications medium and all components attached to that medium whose responsibility is the transference of information. Such components may include, but are not limited to, hosts, packet switches, telecommunications controllers, key distribution center, access control centers, technical control devices, and other components used by the network.

As the definition implies, though we recognize that networks can be viewed as distributed systems and therefore be evaluated at a high service level of abstraction, we chose not to. Basically we believe that though implementation detail does not belong in a set of criteria, viewing the network at too high a level of abstraction would lead the evaluator to ignoring too many details and security problem areas.

4.2 Structure

The TNEC is divided into two major parts. Part I provides network-wide level criteria and is meant to be used for evaluating the security behavior of the network as a whole. Part II provides network components criteria and is meant to be used to evaluate components in isolation just as the Orange Book is used today to evaluate ADP products in isolation. Both Part I and Part II are closely linked to and derived from the Orange Book.

The TNEC classifies networks into four hierarchical divisions of protection. However, as of this writing, the TNEC does not contain "subdivisions" (i.e., classes) as does the Orange Book. This was done for a number of reasons--none of which are sacred. First, it is simpler to do initial analysis if fewer categories of protection need identification. Second, it is not at all clear that even if simplicity was not an objective we could at the early stages of TNEC development, identify additional significant gradations in network security requirements. The divisions are referred to as:

* ND: Minimal Protection
* NC: Controlled Access Protection
* NB: Mandatory Protection
* NA: Verified Design

4.3 Policy

One of the conclusions reached by many of the groups in New Orleans was that a major deficiency of the Orange Book for network evaluations was the lack of policy requirements for transmission integrity and denial-of-service. Though some individuals have been heard to say that integrity is addressable within the context of an Orange Book evaluation, no explicit requirements are stated for integrity in that document. The overall scope of the TNEC is broader than the Orange Book, as the TNEC not only addresses the compromise problem but also addressed transmission integrity and denial-of-service. However, because we recognize that different parts of the network and even different portions of a component within a network will probably be used to meet the three separate sets of requirements, separate evaluation and separate ratings are suggested for each. It is quite possible that a network may meet the NA requirements for compromise

but only achieve an NC for denial-of-service and an ND for integrity. However if these sets of ratings satisy the network design requirements, so be it. The point is, that unlike the Orange Book which only requires one kind of policy implementation, the draft TNEC is meant to meet a number of difference types of security requirements which may and probably will vary in importance to mission objectives.

4.4 The Divisions

As the TNEC is in a state of flux as of this writing, a detailed description of individual criteria may serve only historical purpose. However the following short description of the overall character of each network division may prove useful.

4.4.1 Division ND: Minimal Protection

This devision provides minimal security. There are no security features which are trusted to protect against compromise, integrity, or denial-of-service. This division is reserved for those networks that have been evaluated but that fail to meet the requirements for a higher evaluation division.

4.4.2 Division NC: Controlled Access Protection

This division provides for minimal data compromise, integrity, and denial-of-service protection. Networks within this division are not required to make security decisions based on the level of sensitivity of information being transmitted. Security decisions based on the classification of information are handled administratively.

Instead of the discretionary access control as required in the Orange Book the draft TNEC requires a "Network Discretionary Access Control". At the NC level the network knows nothing about the sensitivity level of data being transmitted, only that hosts and other network components are attempting to communicate with each other or to use functions or services of the network. This criteria requires that the network be able to limit communication between components based on their identity. At the NC level this is the only policy requirement for limiting a subscriber's capabilities on the network.

In the accountability area, the draft TNEC requires all network components to identify themselves to the TNB before service can commence. Identity however will not require authentication for NC networks. There is also a requirement for audit trail maintenance.

4.4.3 Division NB: Mandatory Protection

In this division, the portion of the Trusted Network Base (TNB) that deals with compromise is based on a clearly defined and documented formal security policy model. It requires mandatory access control enforcement over all network resources. This policy is stated in terms of a set of interconnection rules that take into account that all network components must be accredited over some security range, where the range may be as small as a single security level. The rules only allow components to communicate in the range where they share common security levels, and only allows data flow between components communicating at the same security level.

Covert channels are addrressed for NB networks and there are requirements for careful structuring of the TNB into protection-critical and non-protection-critical elements. The TNB interfaces must be well defined and its design and implementation should enable more thorough testing and review. The TNEC at this level also requires Trusted Facility Management and Configuration Management.

4.4.4 Division NA: Verified Design

A network in Division NA must satisfy the reference monitor requirements that it mediate all accesses of subject to objects, be tamperproof, and the distributed portions of the TNB shall be small enough to be subjected to analysis and tests. To this end, the distributed TNB is structured to exclude code not essential to security policy enforcement, with significant systems engineering

during TNB design and implementation towards minimizing its complexity. A distinguishing feature of networks in this division is the analysis derived from formal design specification and verification techniques and the resulting high degree of assurance that the TNB is correctly implemented. This assurance is developmental in nature, starting with a formal model of the security policy and formal top-level specification (FTLS) of the design. Independent of the particular specification language or verification system used, there are five important criteria for Division NA design verification:

* A formal model of the security policy must be clearly indentified and documented, including a mathematical proof that the model is consistant with its axioms and is sufficient to support the security policy.

* An FTLS must be produced that includes abstract definitions of the functions the TNB performs and of the hardware and/or firmware mechanisms that are used to support separate execution domains.

* The FTLS of the TNB must be shown to be consistent with the model by formal techniques where possible (i.e., where verification tools exists) and informal ones otherwise.

* The TNB implementation (i.e., in hardware, firmware, and software) must be informally shown to be consistent with the FTLS. The elements of the FTLS must be shown, using informal techniques, to correspond to the elements of the TNB. The FTLS must express the unified protection mechanism required to satisfy the security policy, and it is the elements of this protection mechanism that are mapped to the elements of the TNB.

* Formal analysis techniques must be used to identify and analyze covert channels. Informal techniques may be used to identify covert timing channels. The continued existence of identified covert channels in the system must be justified.

In keeping with the extensive design and development analysis of the TNB required of networks in Division NA, more stringent configuration management is required and a network security administrator is supported.

5.0 Summary

As of this writing the Center has put a significant effort into the development of Trusted Network Evaluation Criteria. A National forum was organized and held at the DoD Computer Security Center Invitational Workshop on Network Security. Products of that workshop have been used by the Center to formulate draft criteria. The Criteria fall into two types: (a) global criteria, to be used for evaluating the network as a whole; and (b) component criteria to be used for evaluation of individual elements that are to be incorporated in a network and are to play a part in the enforcement of a security policy. In terms of policy this document differs from the Orange Book in that it includes requirements to insure integrity of data transmission as well as requirements to assist in protecting against denial-of-service.

The draft TNEC is about to go out for review by a large divergent group of experts. After receiving their comments we will revise the document and reiterate the process. We expect this process to finally result in the Center's being able to provide guidance in this complex area.

REFERENCES

1. Brand, S. and Arsenault, A. "Network Security Issues " in Proceedings of the Department of Defense Computer Security Center Invitation Workshop on Network Security, DoD Computer Security Center, March 1985.

2. Cerf, V. "Report of the Denial-of-Service Group" in Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security, DoD Computer Security Center, March 1985.

3. Denning, D.E. "Report of the Accountability Group" in Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security, DoD Computer Security Center, March 1985.

4. DoD Computer Security Center, Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83, 15 August 1983.

5. Downs, D. "Report of the Configuration Management and Testing Group (Assurance II)" in Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security, DoD Computer Security Center, March 1985.

6. Kent, S. "Security for Network Components" in Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security, DoD Computer Security Center, March 1985.

7. Lane, J. "Report of the Policy and Models Group or Plato Agonistes" in Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security, DoD Computer Security Center, March 1985.

8. Lipner, S. And Bailey, D. "Report of the Access Controls Group" in Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security, DoD Computer Security Center, March 1985.

9. Rushby, J. "Report of the Working Group on Verification and Covert Channels (Assurance III)" in Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security, DoD Computer Security Center, March 1985.

10. Snow, B. "Report of the Network Architecture Group of the Invitational Workshop on Computer Security" in Proceedings of the Department of Defense Computer Security Center Invitational Workshop on Network Security, DoD Computer Security Center, March 1985.

AN APPROACH TO MULTI-LEVEL SECURE NETWORKS REVISION 1

Leslee L. O'Dell*
Department of Defense
Computer Security Center
Ft. Meade, MD 20755

Introduction

This paper presents an approach to developing a multi-level secure network and the security mechanisms necessary to achieve this. The design will convert an existing network into a multi-level secure network by placing a Front End Unit (FEU) on all hosts and by creating a Central Access Control Center (CACC) which is responsible for mediating accesses between hosts. The design approach influences the philosophy which supports the necessary security requirements. Therefore, this design requires that the needed security requirements be addressed at two levels: the "Network System Level" and the "Component Level."

Presently, network requirements do not exist. In order for this network design to provide multi-level assurances, network requirements need to be defined. The best approach to achieving multi-level security is to identify two distinct sets of security requirements. The first set of security requirements applies to the "Network System Level." On this level, the concern is in guaranteeing secure communications between network subscribers at the appropriate security levels. The second set of security requirements applies to the "Component Level." On this level, the concern is to identify the distributed set of network requirements each component is responsible for enforcing and to identify the security mechanisms the components must provide to ensure secure implementation of the network requirements.

Definitions

The following definitions are a necessity in order to fully understand the network structure and requirements discussed within this paper.

a. The "Network System Level" is defined as all components in the "Component Level" working together as one system to satisfy a defined set of network system security requirements. In order for this to be properly implemented, the requirements for the "Network System Level" are distributed or duplicated among the components.

b. The "Component Level" is defined as the actual physical devices which implement the "Network System Level" security requirements. In this case, the physical devices are the FEUs and the CACC. Each functionally separate component in the "Component Level" enforces its own security properties as well as all, or a subset of, the network system's security properties.

c. A Trusted Network Base (TNB) is defined as the totality of protection mechanisms which are distributed among the network components and which support and enforce the network security policy. These distributed network protection mechanisms are supported and protected by the TCB within each network component.

d. A Security Range includes both the maximal element and the minimal element between two sets of security levels and all security levels falling between the maximal and the minimal. Where the maximal element of a range is the least upper bound of the set of levels in the range and the minimal element of the range is the greatest lower bound of the set of levels in the range. Dominates means the following: classification(a) is greater than or equal to classification(b) and compartments(b) is a subset of compartments(a). Therefore, (a) dominates (b).

Security Requirements

Because the "Network System Level" bears a relationship to an Automatic Data Processing (ADP) system, the "DoD Trusted Computer System Evaluation Criteria" (Criteria), CSC-STD-001-83, forms a reasonable basis from which to develop network system security requirements. The specific network security requirements were developed by extending and tailoring the multi-level security requirements stated in the Criteria. Although the Criteria is the backbone on which the network security requirements were developed, only the extensions made to the Criteria will be discussed in this paper. The following areas had to be extended: Formalisms, Mandatory Access Control (MAC), Discretionary Access Control (DAC), Identification/Authentication (I/A), Trusted Path, Architecture, Labelling and Auditing.

In addition, each network component is concerned with securely supporting its set of multi-level network requirements. In order for the component's operating system to securely manage its set of network security requirements, direction from the component's TCB is necessary. Since the network security mechanisms manipulate

*The ideas and conclusions presented in this paper are those of the author. However, nothing in this paper should be interpretted as representing an official endorsed Government Network Security policy.

multi-level objects and these mechanisms receive direction from the component's TCB, the components are required to provide multi-level assurances. But not all of the multi-level requirements are applicable on the "Component Level" because the components major role is to satisfy the network security and functional requirements. Thus, some of the requirements stated in the Criteria are interpreted for the "Component Level" because components are being adapted to conform to the network environment. These changes are also discussed in the following paragraphs.

Extensions to the DoD Trusted Computer System Evaluation Criteria

The following sections discuss the necessary extentions made to the multi-level requirements stated in the Criteria. These extentions were made to the Criteria to establish multi-level network security requirements.

Formalisms

The extension starts with the Formalisms required for a multi-level secure network system. As stated previously, a secure network is broken into two levels. Because of this philosophical viewpoint, the formalisms must be divided into three specific divisions. There is a division for the "Network System Level," one for the "Component Level" and one which assures that the "Component Level" formalisms combine to satisfy all "Network System Level" requirements (better known as the aggregate demonstration). The following explains these three divisions.

a. The first level of formalisms is the "Network System Level." The "Network System Level" must have a Security Policy, a Formal Security Policy Model (FSPM), and a Formal Top Level Specification (FTLS). These formalisms are needed to adequately define the system's multi-level security properties for secure communications between components.

b. On the component level, formalisms are needed for each functionally different component. Each component has a Security Policy, FSPM, FTLS, and Source Code. The reason for the component having formalisms is two fold:

1. In order for the component to maintain the assurance level of the multi-level network, the component itself must also support multi-level properties. Therefore, each component formally defines its security properties which are needed for that component to maintain the multi-level assurance of the network.

2. Each component is an integral part of an overall secure network system and therefore, supports the overall system's security policy. Because the security of the Network System is distributed among several component, each component reflects in its formalisms its set of network distributed security properties.

c. The third level of formalisms is the aggregate demonstration. This is a convincing argument demonstrating that the combination of the component's FTLSs make up the totality of the Network System's FTLS.

Also, as part of the formal verification process, correspondence between source code and FTLS for each component is shown. This correspondence serves two purposes. First, it satisfies the requirement for correspondence between component FTLS and code. Second, it also satisfies the extended requirement for correspondence between system FTLS and code. The reason for the extension is the "Network System Level" does not have actual code. But the "Network System Level" does have formalisms which are correctly reflected within the components' FTLSs. Therefore, when coding a component to reflect its FTLS, code for the system is also being developed and the need for correspondence between system FTLS and code is established. Then, with the argument that the combination of the components' FTLSs correctly reflects the system's FTLS, a correspondence between the components' FTLSs and code implies a correspondence between the system's FTLS and code.

Mandatory Access Control

The rules governing Mandatory Access Control (MAC) decisions for both the "Network System Level" and the "Component Level" are unchanged. However, there is an area which deserves some explanation. This area is to define what "Subjects" and "Objects" are within the "Network System Level."

In order for MAC decisions to be made on the "Network System Level," network "Subjects" and network "Objects" must be defined. According to the paper "Random Thoughts on Network Security Assurance Issues" by D. Elliot Bell and Marvin Schaefer, "Subjects" are the network subscribers behind the FEUs and "Objects" are the communications channels or "Liaisons." The CACC is responsible for enforcing the accesses of network subjects to network objects. In order for the CACC to make MAC decisions, the CACC must determine what the intersection of the security ranges are between the source network subject and the destination network subject. A Security Range is everyting between, and including, the minimum and maximum security levels a network subject is authorized to handle. This intersection of network source and destination subjects' security ranges identifies the range of security level(s) of network objects(s) which both the source and destination subjects are authorized access to when communicating with each other. However, it is the FEUs responsibility for implementing the network MAC decisions made by the CACC. This is simply a MAC check on each datagram when it is received from either the attached

host or another FEU.

Discretionary Access Control (DAC)

The Discretionary Access Control (DAC) requirement for the "Network System Level" was not extended but in fact rewritten as a System Access Control (SAC) requirement. SAC's responsibility is to define and control communications accesses between "Network System Level" subjects. For instance, SAC mediates accesses between FEUs in the network (e.g., what destination FEUs can the source FEU talk to). It was reasoned that implementing a "Network System Level" DAC would be futile. The difference between DAC and SAC is the DAC controls accesses between subjects and objects whereas SAC controls communication between network subjects.

On the "Component Level," DAC is only required in those components which support a "user." Since FEUs do not support a "user," the FEUs do not have to implement this requirement. However, the CACC supports a System Security Officer (SSO) and a System Administrator (SA) as "users" and therefore must implement the DAC requirement.

Identification and Authentication (I/A)

The identification and authentication (I/A) requirement does not apply to every component level component in a network because not every component has the capability or the need to support a "user." Therefore, only those components on the "Component Level" which support a "user" need to enforce the I/A requirement as stated in the Criteria. In the case of this network approach only the ACC supports a "user." These "users" have been identified to be the SSO and the SA. Thus, the CACC must satisfy the I/A requirement for these "users." However, attached hosts are also considered "users" in this system and the I/A requirement is handled implicitly through hard wired connections. Individual "users" in the Criteria sense are not visible to the FEU.

However, the I/A requirement must be extended within a network system. This extension states that each network datagram must be uniquely identified and authenticated by the TNB. Because the TNB is distributed among the "Component Level," every network datagram is required to be identified and authenticated when it is processed by a component which supports the TNB. Because datagrams are processed by devices that do not support the TNB, it is necessary to protect the integrity of each datagram's I/A information. By enforcing certain key management approaches on each datagram's I/A information, a component's TNB is able to trust received I/A information.

Trusted Path

The Trusted Path requirement stated in the Criteria between the "users" and the Trusted Computing Base (TCB) apply for the standard SSO and SA interface. However, it only applies to those components which support the TNB and have a "user" requirement. Again, in this design approach only the ACC needs to support this requirement.

Also, the Trusted Path requirement had to be extended to include the "Network System Level." A Network Trusted Path is a secure and unambiguous communications path between TNB components. To implement a Network Trusted Path, both source/ destination TNB components need to enforce a key management strategy. By having a keying strategy in place in all TNB components, a TNB component upon receipt of data is able to determine if the path the data traveled was secure and is able to unmistakably identify the communication path. This is a good example of a "Network System Level" distributed requirement that must be implemented by two separate components in order for the requirement to be satisfied.

Architecture

The architecture requirement for both the "Network System Level" TNB and the "Component Level" TCB must be extended because the "Network System Level" requirements are distributed among all "Component Level" components. Because the "Network System Level" TNB is distributed, the architecture of the component's TCB must be modified to include the necessary "Network System Level" TNB functions. Therefore, there will be functions within the component's TCB which operate strictly for the "Network System Level" TNB but are under the control of the component's TCB.

Labels

The labelling requirement is extended on the "Network System Level" to require that each datagram accurately indicate the level of the data. This requirement can be satisfied by using a labelling field within the protocols to indicate the level of the data. However, in a network environment a problem arises in labelling each datagram. This problem, called the granularity of compartments problem, is trying to provide enough positions in the protocol to indicate all the possible compartments available. The point is, a network cannot be expected to support the intersection of all the compartments supported by all the attached hosts. Therefore, it is necessary to define a subset of compartments which were supported and understood by all attached hosts so they could communicate at a more granular compartment level. Another extended labelling requirement is that FEUs in front of untrusted hosts must attach a trusted label which indicates the highest security level that FEU is authorized to process. The reason for this is to have a Trusted label on every datagram.

Auditing

Auditing is an extended requirement

which applies for both the "Network System Level" and the "Component Level." All auditing data is controlled by the "Network System Level" but is a distributed effort among each component in the "Component Level." Hence, each component audits the necessary events for the system and those security relevant events which occur internal to it. Each component is responsible for collecting and protecting the necessary auditing data until the CACC can collect it. It is the responsibility of the CACC to collect, maintain, and protect the aggregation of each component's auditing data as described in the Criteria. It is each component's responsibility to collect and protect the auditing data until it is requested by the CACC. One of the basic reasons for extending this requirement is to have a centralized location where all component and network auditing data can be collected and examined. Some of the things which need to be auditable are the opening and closing of network connections, all security relevant events within each FEU or CACC exploitation of covert channels, and the success or failure of access request.

Conclusion

Again, this paper only presents the requirement extensions which further enhance network security by using the stated design approach. Unless otherwise stated, it is assumed that all other multi-level requirements stated in the Criteria will be satisfied by both the "Component Level" and the "Network System Level." It is presumed that these requirements may not be sufficient for a different design approach.

Other areas of interest are:

a. Data Base Management - There needs to be a data base model which enforces the security policy while retrieving information from the data base.

b. Formal Verification - In the area of formal verification, there needs to be developed an automated method which can assure that the combination of individual component's FTLSs completely reflects the system's FTLS. Also, in the area of formal verification, system level requirements when reflected in components' FTLSs may manifest themselves within the components' FTLSs in terms of different requirements. Because of this possibility, other areas of formal verification are necessary to show correct performance in areas which do not deal with compromise.

c. Protocols - There are several protocol areas which need to be expanded. (e.g., for example, unforeseen potential interactions between protocols which leave the system in a deadlocked or undefined state. Also, tools are needed to help analyze protocols.)

d. Testing - As stated in "Random Thoughts on Network Security Assurance Issues" by M. Schaefer and D. Bell, "much is yet to be learnt about network security testing."

e. Some of the same areas of concern for standard operating systems are concerns also for networks and need to be examined. These areas of concern include ways of dealing with denial of service and ensuring data integrity.

Acknowledgements

This paper would not have been possible if it had not been for Dr. David Bell's foresight and help when the network requirements for this particular project were being developed and tailored. I would personally like to thank Dr. David Bell, Daniel Edwards, Dave Solo, Marvin Schaefer, Steven LaFountain, and Tom Parenty for the many different ideas, and technical discussions which greatly influenced this paper. Finally, a special thanks goes to Janet Swain for her contribution and support.

Determining Security Requirements for Complex Systems with the Orange Book*

Carl E. Landwehr

Computer Science and Systems Branch, Code 7593
Information Technology Division
Naval Research Laboratory
Washington, D.C. 20375

H. O. Lubbes

Computer Resources Division, Code 814T
Space and Naval Warfare Systems Command
Washington, D.C.

1. Introduction

This paper presents a method for determining the hardware and software security requirements of a system, based on

(1) the local processing capability available to a system user;

(2) the kind of communication path between the user's local device and the primary system components;

(3) the flexibility of the processing capability the system provides to the user;

(4) the environment in which the system was developed; and

(5) the difference between the clearance held by the least cleared user of the system and the classification of the most sensitive data processed by the system.

This method can be understood as a risk evaluation of a system that can be conducted at a very early stage in the life cycle of a system and repeated as the structure and functions of the system change during its development and operation. Depending on the inherent risk that a system (or system design) displays, different levels of security requirements may be imposed in order to reduce the operational risk of the system to an acceptable level. Applications of this method to several environments are provided as examples. Although developed based on consideration of DoD environments, the method seems applicable to other environments to the extent that the Orange Book requirements apply to them.

The technique described here does not consider requirements for degaussing of removable storage, TEMPEST requirements, protection from physical hazards, emergency destruction, or other security requirements not related to the hardware and software architecture of the system.

2. Structure of the Orange Book

The DoD Trusted Computer System Evaluation Criteria (the "Orange Book" [1] provides a set of security requirements of two kinds: specific security feature requirements, which call for particular system functions in order to provide data security, and assurance requirements, which call for testing, documentation, and verification to assure that the security features are correctly implemented. A system that satisfies all requirements listed in the Orange Book would be designated A1. Systems that satisfy specified, nested subsets of the requirements are designated B3, B2, B1, C2, C1, D, in order of decreasing requirements.

The Orange Book does not provide guidance as to what level of system is appropriate for a particular operational environment. A draft application doctrine [2] has been developed, however, that defines the level of system required for a particular environment based only on the classification of the data processed by the system, the clearances of its users, and the environment in which it was developed. This simple scheme is inadequate for use in assessing security requirements of many complex systems; a more comprehensive method is proposed below.

3. Applying Technical Computer Security Guidance Effectively

Although it is imperfect in many respects, as a technical basis for specifying computer security requirements, the Orange Book is the most comprehensive and current document available. A method is needed for applying the Orange Book to the components of large scale, geographically dispersed systems, so that the appropriate requirements from the Orange Book book can be identified for each host system. Such a method is defined below. As shown in Figure 1, it involves:

(1) extracting from each system (or system design) the factors that affect the risk that its operation may lead to the unauthorized disclosure of sensitive information,

(2) quantifying these factors, and

(3) determining system security requirements (in terms of the levels defined in the Orange Book) that reduce the system risk to an acceptable level.

This method can be understood as a risk evaluation based on the threat of unauthorized disclosure of sensitive information. The asset of the system is sensitive information, defined in terms of its classification level, and the vulnerabilities of the system depend on the degree of control it exerts on its users. The system risk combines the value of the assets, the vulnerabilities of the system, and the clearance of the users.

Identifying the Risk Factors

To determine a system's security requirements it is necessary to consider the environment in which that system operates. The Orange Book specifies levels of requirements independent of system environment; the draft application doctrine [2] characterizes a system's environment in terms of three parameters: the maximum clearance of the least cleared user, the maximum classification of data processed by the system, and the environment in which the system is developed and maintained (open or closed). While simple to evaluate, these parameters omit important factors that affect actual system risk.

The following paragraphs explain the factors that should be taken into account. For each factor, different levels of risk

* The technique presented here represents the technical judgment of the authors and does not necessarily represent the views or policy of their respective organizations or of the U.S. Navy.

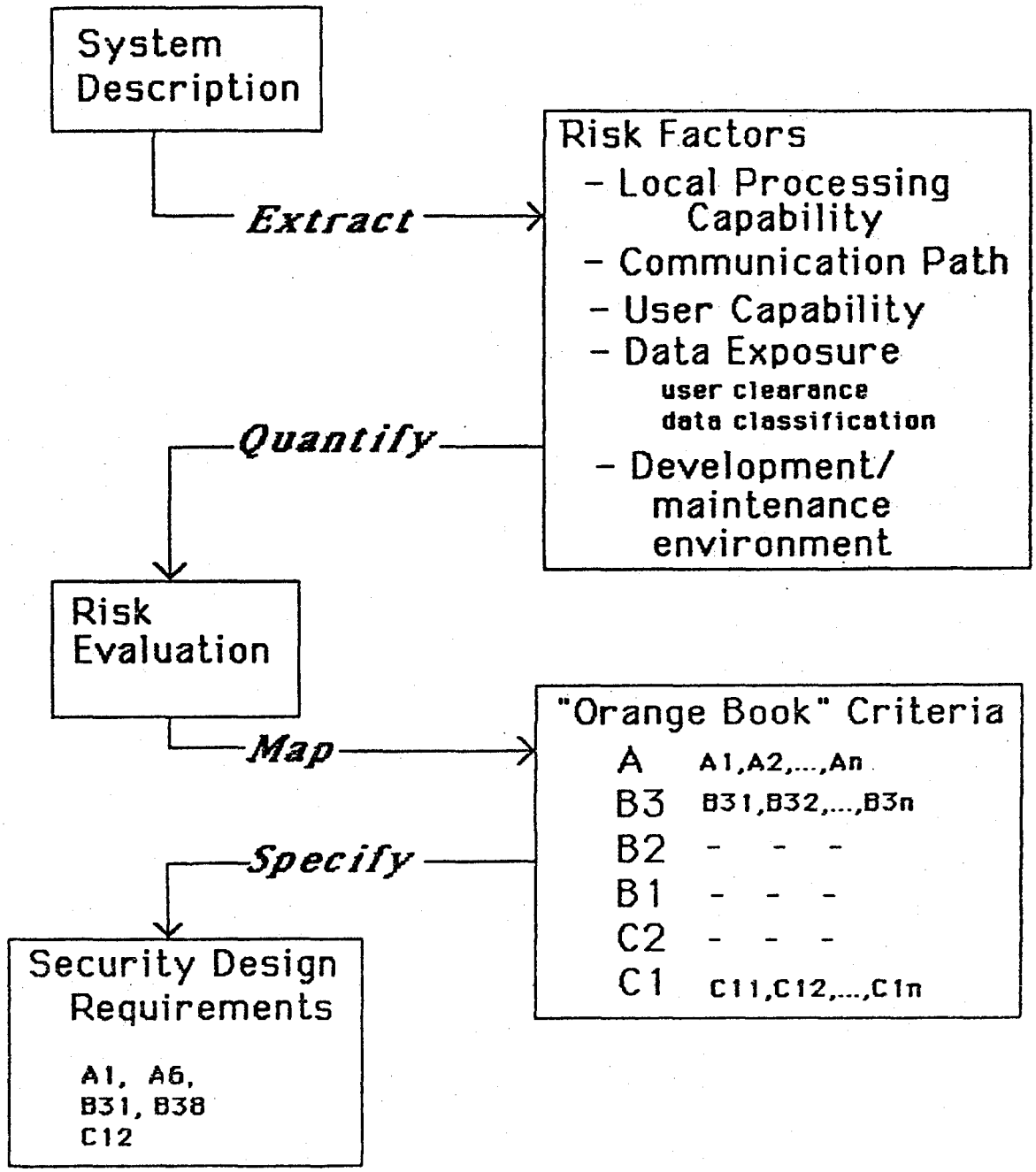


Figure 1. *Steps in applying guidance.*

are defined so that the difference between two adjacent levels in each factor represents a roughly comparable increase (or decrease) in risk. Factors are defined so that they are roughly independent -- a change in one factor does not imply a change in another factor. These properties allow numbering the risk levels and combining them in most cases using simple addition.

Something as abstract as risk cannot be quantified precisely. Recognizing this, we have not attempted to make fine distinctions, and no doubt some systems will still fall near the boundaries of the proposed classes. Nevertheless, the scheme described below, coarse as it is, captures the intuition and experience of computer security practitioners and is preferable to simply setting these considerations aside because they cannot be made precise.

Local Processing Capability. Some systems have receive-only terminals (e.g., stock transaction displays, airline terminal monitors); users of such terminals have no way to enter system commands directly. Such terminals represent a lower level of risk than typical interactive terminals that permit both sending and receiving information. Replacing a fixed-function interactive terminal with a programmable terminal, personal computer, or other programmable device would introduce a still higher level of risk, since the user can now program his terminal to enter commands for him. A user who accesses a system from a fixed-function terminal but via a programmable host computer would be considered to have the same local processing capability as one who uses a personal computer as a terminal. The identified risk levels for local processing capability are:

Level 1: receive-only terminal

Level 2: fixed-function interactive terminal

Level 3: programmable device (access via personal computer or programmable host)

Communication Path. The communication path between a terminal and host can also affect system risk. A terminal that has a simplex receive-only link to its host via a store-and-forward network (e.g., via radio broadcast) poses less risk than one that is connected via a duplex store-and-forward link, since the simplex path prevents the user from submitting requests to the system. Terminals that are connected to a host either directly, through a local-area network, or long-haul packet network (e.g., Telnet, DDN) offer increased possibilities for penetration and misuse (inadvertant or otherwise) over those connected only through a store-and-forward net because of the increased bandwidth and closer host-terminal interaction they permit. The identified risk levels for communication path are:

Level 1: store/forward, receive-only

Level 2: store/forward, send/receive

Level 3: interactive, via direct connection, local-area net, or long-haul packet net

User Capability. Regardless of the local processing available to a user or the communication path he uses to access a host, if that host is programmed only to provide predefined outputs regardless of the inputs the user presents, it is less risky than a system that responds to user transactions. In this sense, the system that generates the ticker tape for a stock exchange is less at risk to the terminals that display the tape than an interactive electronic banking system is to automated teller machines. Finally, a transaction-based system is less at risk from its users than a system that permits its users full programming capabilities. The identified risk levels for user capability are:

Level 1: output only

Level 2: transaction processing

Level 3: full programming

Development/Maintenance Environment. A system that has been developed and is maintained by cleared individuals under close configuration control (closed environment) should pose less risk than one that is not developed and maintained in this way (open environment). This distinction has been proposed in the draft application doctrine [2]. It seems a reasonable one, but relatively few examples of systems developed and maintained according to the proposed definition of "closed environment" have been identified outside of the intelligence community. For simplicity, we assume that systems are developed and operated in an open environment. Systems that are developed and maintained in a closed environment may therefore be subject to slightly less stringent requirements than will result from our approach.

Data Exposure. A system that has a greater disparity between the clearance of its least cleared user and the classification of the most sensitive data it processes is more at risk than one that has a lesser disparity. The draft application doctrine proposes a scheme for numbering and classifying "risk range" we adopt this scheme but call it "data exposure" to distinguish it from other risk factors. Although clearance and classification levels used are based on the DoD system, they do include levels for sensitive but unclassified data and users authorized access for such data. For non-DoD environments, it seems likely that analogous clearance/classification levels could be defined. Clearance levels are identified as:

Level 0: uncleared

Level 1: uncleared, but authorized access to sensitive unclassified information

Level 2: confidential clearance

Level 3: secret clearance

Level 4: top secret/background investigation

Level 5: top secret/special background investigation

Level 6: top secret/special background investigation, with authorization for one compartment

Level 7: top secret/special background investigation, with authorization for more than one compartment

Classification levels are numbered:

Level 0: unclassified

Level 1: sensitive unclassified information

Level 2: confidential

Level 3: secret

Level 4: secret with one category

Level 5: top secret with no categories or secret with two or more categories

Level 6: top secret with one category

Level 7: top secret with two or more categories

Data exposure is computed as the difference between the level of the least cleared user of a system and the maximum level of data processed by the system. It thus ranges from a value of 0 (all users cleared for all data) to 7 (system processes top secret data with two or more categories and some users are uncleared).

Applying the Risk Factors

For a particular system, each of the risk factors needs to be evaluated in order to assess the overall ("system") risk. With minor exceptions, the system risk is simply the sum of the risks of the individual risk factors. Based on system risk and data exposure, security requirements can be determined. These requirements are characterized here in terms of the levels defined in the Orange Book because they have been published and reviewed widely. If a different subsetting of the Orange Book requirements later proves more appropriate than the current set of levels, the new subsets can be substituted. Tables 1-3 provide the necessary mappings between factor values, risk factor levels, and security requirements. The first two tables are only needed because of the exceptions mentioned above; usually, Table 3 can be used directly with the sum of the individual risk factors.

Note that in a given system, different terminals may provide different functions, lead to different levels of risk, and impose different security requirements. Security requirements for the system as a whole must be determined on the basis of the most risky part. As noted previously, the tables below assume all systems are developed/maintained under conditions of an open environment.

Table 1. Together, local processing capability and communication path characterize what computer security literature refers to as the "process coupling" risk. This term is intended to cover how well a process in one computer can maintain its integrity in the face of attempts to subvert it from outside. A high degree of coupling represents a close degree of interaction between two processes, and hence a greater vulnerability of one to the other. If there is a very limited, well-defined set of requests one process can make of the other, then the degree of process coupling will be low. Process coupling risk in a system, as shown in Table 1, is the sum of the local processing capability and communication path risks, with one exception. A fixed function interactive terminal attached to a one-way store-and-forward communication path does not increase risk over a receive-only terminal on the same link. A programmable device increases risk over the interactive terminal, since, if improperly programmed, it might corrupt labels transmitted with data.

Table 2. The process coupling value from Table 1, combined with the appropriate user capability factor value yields an overall system risk independent of the data exposure. As in Table 1, the entries of Table 2 have been obtained by summing the risk factor values from each axis. The entries for a process coupling of 2 (receive-only or interactive terminal on a

Table 1. Process Coupling Risk

Local Processing Capability	Communication Path		
	1. S/F Net (one-way)	2. S/F Net (two-way)	3. I/A Net or Direct Connection (LAN,DDN)
1. Receive-only Terminal	2	3	4
2. Interactive Terminal (fixed function)	2	4	5
3. Programmable Device (Access via personal computer or programmable host)	4	5	6

Table 2. System Risk

User Capability	Process Coupling Risk				
	2	3	4	5	6
1. Output-only (Subscriber)	3	4	5	6	7
2. Transaction Processing	-	5	6	7	8
3. Full Programming	-	6	7	8	9

Table 3. Mapping System Risk and Data Exposure to Orange Book Levels

Data Exposure	System Risk						
	3	4	5	6	7	8	9
0	C1	C1	C1	C1/C2	C2	C2	C2
1	C1/C2	C2	C2	C2	C2/B1	B1	B1
2	C2	C2/B1	B1	B1	B1	B1/B2	B2
3	B1	B1	B1/B2	B2	B2/B3	B3	B3/A1
4	B2	B2/B3	B3	B3/A1	A1	A1	A1
5	B3/A1	A1	A1	-	-	-	-
6	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-

receive-only link) have been omitted for user capabilities of transaction processing and full programming, since a receive-only link cannot support either of these capabilities.

Table 3. This table relates the system risk with the data exposure to yield a level from the Orange Book that defines the security requirements for the system. As noted above, the Orange Book levels may later be replaced by related, but distinct, sets of features and assurances. The entries in this table were generated by working through examples and considering the guidance provided by the draft application doctrine [2] and current DoD directives governing compartmented mode. Blank entries indicate that, for the specified data exposure level and system risk, it appears technically infeasible to meet the appropriate security requirements at the time.

4. Examples

A Sea Surface Surveillance System (S4)

Consider the application of the technique outlined above to a hypothetical system that keeps track of objects on the surface of the seas (see Figure 2). The system collects information from a variety of open and secret sources and distributes it to a variety of customers. The system maintains a data base of sighting information that is both automatically and manually updated. There are two major classes of users: analysts and subscribers.

S4 analysts are the direct operators of the system: they are called on to resolve ambiguities when the system cannot associate a particular sighting with a particular platform, they can cause messages to be sent to subscribers automatically on a regular basis, and they can update the data base. They operate interactive terminals that are located in S4 spaces and connected directly to the S4 computers.

S4 subscribers are the recipients of reports generated by S4. They are located outside the S4 spaces and receive reports over a variety of different communication networks on receive-only terminals. They cannot directly enter data into the S4 system, but they can issue requests (via normal message channels) for regular updates on the location of particular objects, for example. These requests are received by S4 analysts who cause filters to be set up that automatically channel relevant reports to the subscriber. Once the appropriate filter is set up, no further human intervention is required.

Since analysts and subscribers are permitted different kinds of functions, have different clearances, and communicate with the S4 system over different paths, it is necessary to apply this technique to each class of user separately.

Local Processing Capability. Analysts operate fixed function interactive terminals, so they represent a risk level of 2. Subscribers operate receive-only terminals, yielding a risk level of 1.

Communication Path. Analysts communicate with S4 machines directly, so their risk level is 3. Subscribers communicate over a one-way store-and-forward network, so their risk level is 1.

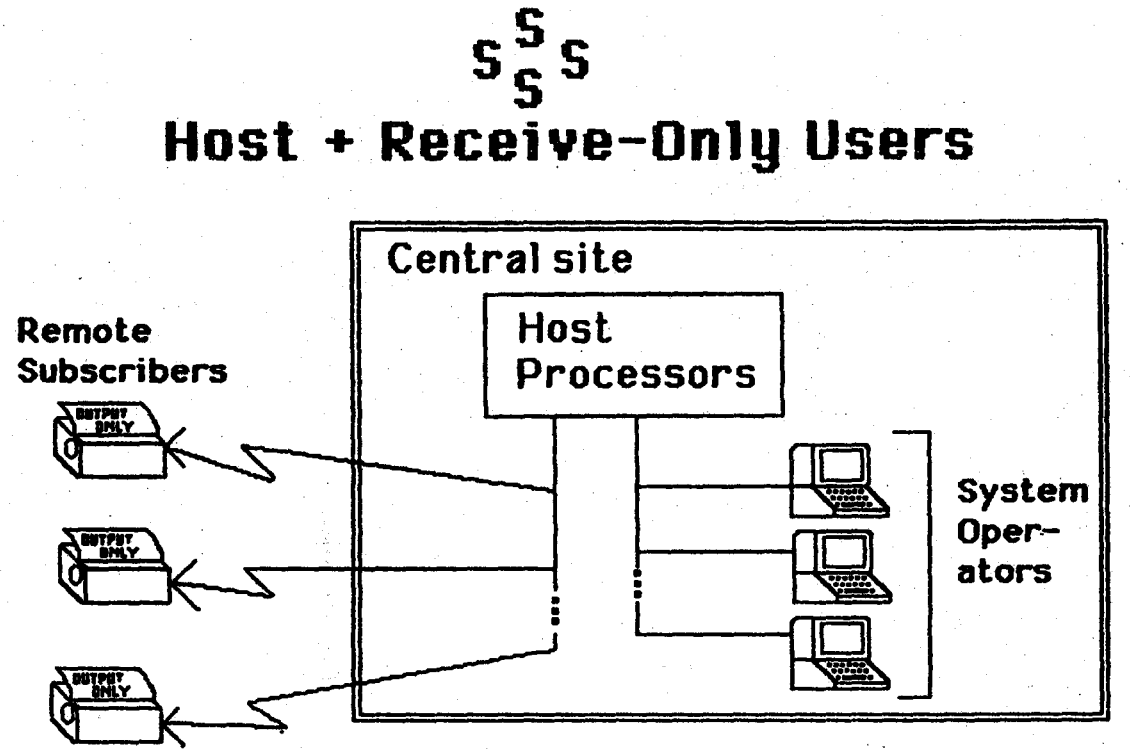


Figure 2. *Original S4 system.*

User Capability. Analysts are permitted to issue transactions directly to S4, but they do not have full programming capability, so the risk level is 2. Subscribers have output-only capability, so the risk level is 1.

Data Exposure. S4 processes data at the TS level with multiple compartments, so the classification level is 7. S4 analysts hold TS clearances with SBI and are authorized access for all compartments that S4 processes. Consequently, their clearance level is also 7 and the data exposure for analysts is 0. Some S4 subscribers hold only Secret clearances with no compartment authorizations, so their clearance level is 3, yielding a data exposure for subscribers of 4.

Using the Tables. First, for analysts, Table 1 shows that a local processing capability risk of 2 and communication path risk of 3 yields a process coupling risk of 5. Table 2 combines a user capability risk of 2 with a process coupling risk of 5 to yield a system risk of 7. Table 3 maps a data exposure of 0 and a system risk of 7 to a C2 level system requirement.

For subscribers, Table 1 combines a local processing capability risk of 1 with a communication path risk of 1 to yield a process coupling risk of 2. Table 2 combines a user capability risk of 1 with a process coupling risk of 2 to give a system risk of 3. Finally, Table 3 maps a data exposure of 4 and a system risk of 3 to a B2 level system requirement.

Since S4 includes both kinds of users, the more stringent of the two requirements (i.e., B2) would apply. Changes to the environments of either subscribers or analysts (such as the introduction of personal computers in place of fixed function terminals) would require the risk evaluation to be repeated, and could lead to a change in the level of security requirement.

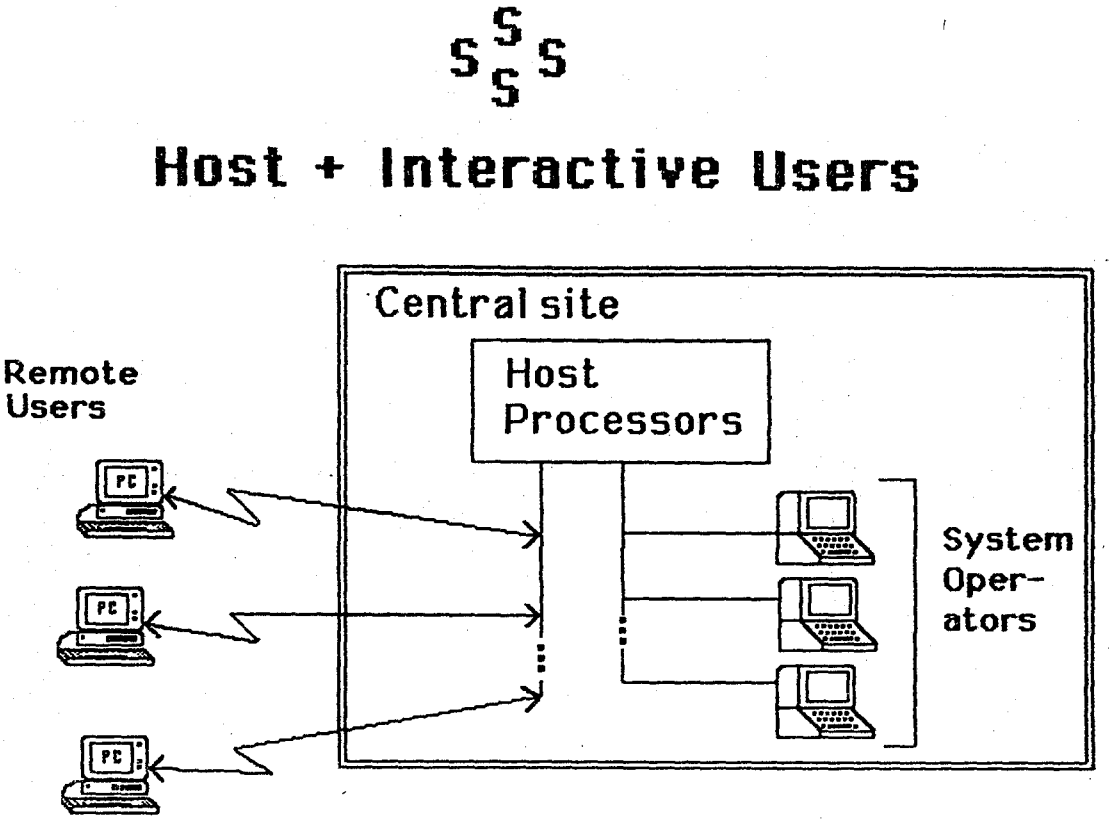


Figure 3. *Evolved S4 system.*

Evolution of the S4 System

Suppose that after initial deployment of S4, its subscribers clamor for terminals more up-to-date than the original receive-only ones. The system sponsor proposes to replace them with personal computers (see Figure 3). What are the effects on the security that the host system needs to provide? The local processing capability risk factor changes from 1 to 3, and the system risk becomes a 5; the data exposure for subscribers is unchanged. Table 3 shows that the host should security should be upgraded from B2 to B3. If, in addition to the personal computers the sponsor permits subscribers to communicate with the system over a real-time network and to initiate transactions, the system risk becomes 8, and an A1 host would be indicated. By estimating the additional cost of replacing or upgrading the S4 host to the B3 or A1 level, the sponsor can quantify the cost of providing new functions while maintaining an acceptable level of risk for the system.

"Orange Book Environment"

The Orange Book does not explicitly define an environment. However, the predecessors of the Orange Book criteria were first developed in the context of an interactive computer system that provided users with directly connected, fixed-function terminals and full programming capability. The corresponding entries in Tables 1 and 2 yield a system risk of 8. Since no data exposure is defined for the Orange Book environment, it is appropriate to consider the result for the Air Force Data Services Center (AFDSC) Multics environment, which provides full programming to users at fixed function, directly connected terminals. AFDSC Multics includes non-compartmented data classified up to top secret and some users have only secret clearances, so the data exposure is 2, and the resulting security requirement from Table 3 is for a B1/B2 system. Multics is currently under evaluation by the DoD Computer Security Evaluation Center and is expected to achieve a

B2 rating.

5. Discussion

Here we address some possible objections to the approach described above.

Objection: the proposed scheme imposes different requirements on a host computer based on characteristics of the user's terminal and the communication path between the terminal and the host. These are outside the security perimeter of the host and therefore should not affect the security required of it.

Response: security considerations include not only protecting data up to the point that it leaves the system but also resisting attacks on the system mounted by external users. Users with personal computers and direct connections to systems have proven a greater threat (e.g. in terms of their ability to defeat password schemes) than those who have only fixed-function terminals at their disposal. Each higher Orange Book level adds assurance requirements as well as security feature requirements. While the security features added at a particular level may or may not improve protection against threats posed by terminals and networks connected to a host, the increased assurance provided by each incremental level should decrease the likelihood of flaws that could be exploited from outside the security perimeter. It is thus appropriate to increase the Orange Book level required of a host based on the risk factors assigned to the user capability and communication path.

Objection: the proposed approach in some cases permits hosts to meet lower security requirements than would the draft application doctrine[2].

Response: the approach proposed here distinguishes aspects of application system structure that reduce its vulnerability to outside attacks. The draft application doctrine determines the level of system required based primarily on the clearances of system users and the classification of data stored in the system. There is no distinction, for example, between a system in which users can only view output and one in which users can construct and execute their own programs. Consequently, the proposed requirements must be based on the worst case assumption (user programming). By providing a more detailed model of the environment, the approach proposed here permits a more accurate assessment of the security actually required.

Objection: previous attempts to distinguish rigorously between a system that can be programmed and one to which only transactions can be submitted have failed.

Response: while a formal mathematical distinction between systems that users can program and those that perform a fixed set of functions in response to user requests may never be defined, it does not seem to be a difficult distinction to make in practice. In cases that are difficult to decide (e.g., a "transaction-processing" database system that permits a complex query and update capability) it is safe to assign the system the higher risk factor.

Objection: because the proposed approach determines host security requirements partly based on system architecture, changes to the architecture may lead to different security requirements.

Response: this is actually a benefit of the approach. As a system changes during its design, development, and operation, the effects of those changes on host security requirements can be easily assessed, providing a practical way to use the Orange Book requirements throughout the system life cycle. If, for example, a B2 host will not be available to support an application as originally planned and a B1 host must be used instead, the approach proposed here can help determine how system functions, user capabilities, or communication paths could be restricted to compensate for the less secure host. Conversely, if new functions or terminals are added to a system already under development, this approach can indicate whether host security will need to be upgraded as a result. The only tradeoff that would be recognized under the draft application doctrine would be to limit the classification of the data to be processed by the system or increase the clearance of its users.

6. Conclusion

We have presented a scheme for determining an appropriate set of host security requirements using the requirements and levels identified in the Orange Book. The scheme takes into account the functions available to a user locally, the communication path used to gain access to the host, and the functions the host provides, as well as the user's clearance and the classification of data processed by the host. By including these system characteristics, this technique makes it possible to assess trade-offs among system function, system architecture, and system costs while maintaining an acceptable level of system risk.

References

1. Department of Defense Trusted Computer System Evaluation Criteria, DoD Computer Security Evaluation Center, CSC-STD-001-83, 15 August 1983.
2. Brand, S. Environmental Guidelines for Using the DoD Trusted Computer System Evaluation Criteria. Proc. Seventh DoD/NBS Computer Security Initiative Conference, Sept., 1984, Gaithersburg, MD, pp. 17-23.

☆U.S. GOVERNMENT PRINTING OFFICE:1985 529 165 30939