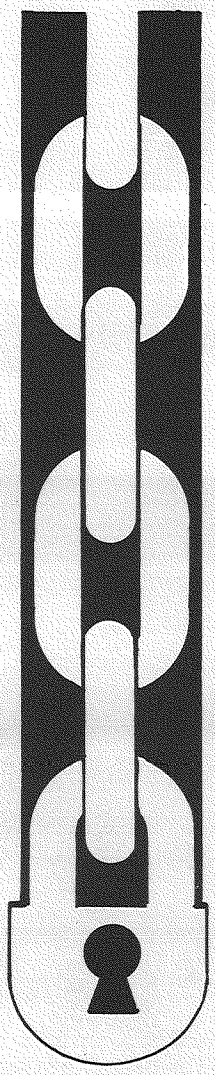


7th DOD/NBS Computer Security Conference

September 24-26, 1984

TABLE OF CONTENTS

OPENING REMARKS

ROBERT L. BROTZMAN

DIRECTOR, DoD COMPUTER SECURITY EVALUATION CENTER

I'd like to join Jim Burrows in welcoming you to this, my first Computer Security Initiative Conference here at NBS. As Marv mentioned, I assumed responsibility for the Center in mid-July.

Mine is not the only new face to appear in the Center since last year's conference. For openers, the Center has grown 40 percent from the 114 we had when Mel Klein spoke to you last November to our present staff level of about 160. Much of the Center's personnel growth was to continue staffing a fledgling organization that began life with a lot of existing pre-Center DoD responsibilities, including evaluating trusted commercial products and operational systems, specifying the computer security requirements for selected trusted system procurements like the I-S/A AMPE and, of course, writing and publishing the Orange Book, the Evaluation Criteria.

But as industry, the DoD components and other Federal Agencies developed greater security awareness and became more familiar with the Center, requests for our participation in their activities also grew rapidly. In addition to these conferences at NBS, we have moved forward in our education role. We have conducted classes, often with the support of experts from the private sector and other parts of the Government, on: computer security policy, evaluation and certification, formal specification and verification, trusted system technology, and a special senior-level course on computer security awareness for government executives. The Center gives several customized briefings every week to government organizations on classified and unclassified topics in computer security.

At last year's conference, Dr. DeLauer told you that the Center would be conducting an extensive study on the DoD's current computer security posture and its future requirements. That study has been completed and has been distributed for coordination and comment within the Department. The study has already resulted in a close and healthy working relationship between the Center and the Computer Security Focal Points from the DoD Components, particularly now that the Criteria are being incorporated in procurement specifications across the DoD.

You may also recall from last year that both Dr. DeLauer and John Lane remarked that they had "heard it said that a DoD computer acquisition has never been won or lost on the basis of security features." Dr. DeLauer went on to say that "we need to turn that perception around so that our friends in industry can know that we are really serious about security." Well, I can certainly tell you that we are. Several recent DoD procurements

have now called for products that fully satisfy the requirements for specific Criteria classes. And, although I can't go into the details, one DoD organization reported that they have disqualified a major vendor because his proposal did not respond to the evaluation class requirements designated for their ADP procurement. The lesson is clear. The DoD services and agencies are taking the Criteria and computer security seriously in their planning.

We have seen a good deal of positive results from the Center and from the private sector this year. The Evaluated Products List now has its first trusted product entries. I am happy to report that evaluations have been completed on IBM's RACF/MVS Facility in the C1 Class, and SKK's ACF2/MVS Package has fully satisfied the C2 requirements. Other product evaluations are being completed and should be published in the next few months. Among these is the Honeywell SCOMP evaluation. As you know, this is the first evaluation the Center has undertaken for an A1 product. Penetration testing on the SCOMP was recently completed, and the long and suspenseful ordeal is essentially over. There are a few loose ends in the documentation area to tie up right now, but I take real pleasure in telling you that all of the technical requirements in the Criteria have been satisfied by the SCOMP. Both the SCOMP evaluation and the Honeywell MULTICS/AIM evaluations will be published soon.

I am also able to tell you that we are moving forward with the transfer of the formal verification tools to the ARPANET. The developers of FDM, HDM and GYPSY are all under contract to transfer their verification systems to the Center's MULTICS/AIM systems. We expect to be offering access and training in SDC's formal Development Methodology in the near future. Once the transfers have been completed, the Center will also provide access and training in the Hierarchical Development Methodology and the GYPSY Verification Environment.

Many of you were at Conferences here in Gaithersburg when the BLACKER system concept was discussed by the DoD Director of Information Systems. The Center has been assigned responsibility to produce BLACKER as a system that satisfies the A1 requirements. As a network security application, this program is a challenge to all of us, since the Network Security Evaluation Criteria have not been completed. A panel session this afternoon will discuss a number of relevant network security criteria issues, and may give you some idea of the technical consideraitons germaine to these applications.

In assuming this new responsibility, the Center has gone through a reorganization. We still have our original four offices of Product Evluations and Standards, Application Systems Evaluations, Research and Development, and Technical Support. In addition, Dave Bitzer has joined the Center and serves as the BLACKER SPO.

My last news item pertains to myself and the Deputy Director of the Center. Many of you know that Mel Klein retired from the Center following 36 years of service to the Agency, and Colonel Roger R. Schell retired from the Air Force following the completion of his tour with the Center. Mel and Roger built the Center from its beginnings to its current position of technical expertise and service. Our new Deputy Director, Colonel Joe Greene and I have been very busy working with the challenges and opportunities we face in the future. Joe will be speaking to you following this morning's break, so I will not take the time to tell you how capable he is and how fortunate we are to be able to draw on his talents.

I am looking forward to working with you all in the future. I hope you enjoy and profit from this, the Seventh Computer Security Initiative Conference.

KEYNOTE ADDRESS

HONORABLE DAN GLICKMAN

CHAIRMAN, SUBCOMMITTEE ON TRANSPORTATION, AVIATION AND MATERIALS
COMMITTEE ON SCIENCE AND TECHNOLOGY
U.S. HOUSE OF REPRESENTATIVES

I am delighted to be here and especially pleased to see the level of participation in the 7th Conference on Computer Security sponsored jointly by the DoD Computer Security Center and the NBS Institute for Computer Sciences and Technology. The subject of computer security hopefully has emerged from the world of the specialist and entered into the consciousness of the public.

Computers and the communications links that connect them pervade our society. Yet, average citizens seldom realize how computers intrude upon their lives and, further, how much more intrusive computers will become with each passing day.

Possibly for this reason we have also been largely unaware of the possibilities for the improper use of computers or unauthorized access to the information they contain. Even professional computer people have frequently underestimated the threat to their systems, often failing to take the most elementary precaution -- the electronic equivalent of locking the door. Until recently, it was only a few voices in the wilderness who even discussed it at meetings such as this.

Now, however, with the media attention paid to "computer hackers," and the popular films, "War Games" and "Superman III," the public is finally becoming aware that George Orwell may have been right. There really is a "Big Brother" -- although he is not restricted just to the government as in the novel. Intruders into our personal lives in 1984 can be almost anyone with even rudimentary knowledge who seeks to make mischief, gain an unfair advantage, or commit a crime with computers.

Since my subcommittee began its examination of computer and communications security and privacy over a year and a half ago, we've found an incredible level of interest. So I can tell you the public is beginning to hear the message. And your presence here today confirms it.

My interest in this issue came about because a Science and Technology Subcommittee which I chair has jurisdiction over "communications research and development." That interest was personalized last year when I was ordering tickets for an upcoming Orioles game, using a system they have which uses the push buttons on your telephone. And it occurred to me that here was a system ripe for abuse. So I suggested to my staff and the Library of

Congress -- many of you probably know Louise Becker who was until recently the Library's expert on computer security --that we look into it.

After a few weeks of digging around, we found that there was indeed enormous potential for abuse, not only of this system but of all kinds of computer-based information systems. We also found a subject of immense complexity. There is an R&D component to be sure, but we quickly found there are great policy, Federal leadership and national security issues as well. We also discovered that many of these issues were interrelated. Trying to organize them into manageable units was a little like trying to put socks on an octopus.

As many of you know, the Subcommittee held hearings last fall aimed at covering more comprehensively than ever before all of these related aspects.

We began by looking at the full range of threats and vulnerabilities to computer systems. Our first witness was Neil Patrick, a 17-year old high school student, whose exploits as a computer hacker had just drawn national media attention, thereby giving our hearing a timely boost.

He testified that his interest in computers began with an introductory course in the seventh grade. From there he learned various computer languages and using his family's home computer managed to gain unauthorized access to systems around the country, such as the Security Pacific Bank.

He told the Subcommittee that access to government and private-sector computers was often made possible by poor control of passwords. He described how he and other hackers exchanged information on how to break in to computer systems by using electronic bulletin boards.

He explained that hackers seldom see their activities as harmful. The perception that "no harm intended is no harm done" typifies the computer hacker mentality. The same individuals would never consider breaking into someone's home, but destroying an elderly woman's bank record is somehow merely an intellectual challenge.

In fact, when asked when he first questioned the ethical propriety of what he was doing, Mr. Patrick replied, "When the FBI knocked on my door."

And yet, despite all the attention he got, Mr. Patrick clearly illustrated a point that is often overlooked by the media who usually portray such youngsters as the modern-day equivalent of Robin Hood. And that point is that the far greater danger comes not from hackers, but from insiders -- those that already have authorized access.

We received testimony that the great bulk of computer-related criminal actions in the commercial sector are perpetrated by an individual who was authorized to interact with the system and who knew enough about it to exploit it for personal gain.

Furthermore, there is generally little attention paid to establishing the trustworthiness of individuals in critical and sensitive positions. Some corporations do effectively nothing to assure the trustworthiness of critical individuals; others take the minimal step of requiring that employees are bondable; and very few, if any, perform comprehensive background investigations or require specific training to sensitize their employees to problems in computer security.

To deal with this situation, the Subcommittee concluded that both the private sector and the federal government should strengthen clearance procedures for all workers handling sensitive, non-national security data. As part of their training, they should receive awareness training on computer abuse, including the penalities for illegal activities. In addition, automated information systems and related documentation should contain explicit warnings regarding unlawful activities or abuse.

Another area that the hearings were designed to address was the effectiveness of leadership by the federal government. And here we sought to examine several aspects of the government's responsibility. The first of these was its performance in protecting the systems used internally by virtually every agency to carry out its mission.

What we found generally was a depressing picture of drift and inattention to a potentially serious problem. No doubt, a few agencies are well aware of the threats to their systems and are giving high-level management attention to the need for better security. But in the vast majority of cases, the data kept in government computers is susceptable to manipulation, penetration and unauthorized access. And what's worse -- no one seems to be in charge. The administration's approach is basically to ignore the problem.

One specific failure is the lack of training to sensitize both computer and management personnel to the problems in computer security and to teach good security practices.

To achieve this, I am contemplating legislation early next year that would require agencies to provide such training. I also believe the government needs a focal point -- perhaps an institute within an existing agency or even totally separate -- for conducting this training and for assisting agencies in selecting tools and techniques to protect their computers.

Another important element of federal responsibility, I feel, is the need to secure major, national systems that are computer based. And I would argue that this responsibility includes not

only federal systems, but also certain private sector systems that are key to the functioning of our society and which therefore contribute to the "national security" in a broader sense.

For example, the federal government operates the air traffic control network, a massive system designed to keep some 15,000 daily commercial flights from colliding with each other -- an absolutely crucial function. It is a highly automated system and will become even more so when the National Airspace System Plan is implemented. The potential consequences of unauthorized intrusion are enormous.

Clearly the government must assure the security of the ATC system from such misuse. But what about others, such as the electric power grid, or the banking system? Obviously these are private, for the most part, but the country is highly dependent on them.

For this reason, I believe the federal government should assume some level of responsibility for their protection. But no one seems to be worrying very much about it. And I don't find a whole lot of concern even about the federally-operated systems.

I'm convinced there could be several potential disasters out there just waiting to happen. A disgruntled employee, a terrorist group, or someone with criminal intent could easily disrupt or even cause irretrievable damage to our nation. So we can't allow the present state of indifference to continue.

The Subcommittee concluded there is an immediate need to pinpoint responsibility for security of critical national systems and to implement necessary controls. Such a focus would require input from a number of governmental areas (including national security, intelligence, law enforcement, emergency management, and commerce) as well as the private sector, to assure that all facets are addressed.

Another area of federal responsibility that the Subcommittee considered was our role in conducting research and developing technology for use both by the government and in the private sector. Here we found that system managers are often reluctant to invest in technologies that they see as being too costly and which may hamper performance or limit use. What seems to be needed -- aside from greater awareness of the threats -- is research and development of low-cost, effective computer and communications protection equipment.

I believe a cooperative effort among vendors, users and the government is needed to identify areas that might benefit from additional research effort and to channel available resources accordingly. At the federal level, a permanent task force, consisting of both central management and mission agencies, should be established to coordinate computer and communications research

(at the DoD Computer Security Center, the NBS Institute for Computer Science and Technology and NSF).

Turning now to the area of computer crime, the Subcommittee was interested in examining the dimensions of this type of crime along with the need for better laws to deal with it. Of course, this is a tremendously complicated subject and we were only able to scratch the surface. But there was substantial agreement on several points.

The first relates to the lack of adequate definitions and, as a result, of valid statistics on the size of the problem. The only way we can characterize it, is through case-by-case empirical analysis of reported cases. And those probably represent only a fraction of the actual cases because of reluctance of victims to reveal their losses.

As an example of how technology outpaces the law, the GAO pointed to The Crime Control Act of 1968. That law used the qualifying terms "aural acquisition" -- or acquired by use of the ear -- to define interception. As a result, only interceptions by aural means are illegal under this Act. Anyone can conduct unauthorized non-aural wiretapping of data communications without a court order and not be in violation of this law.

Another point of more or less uniform agreement was the need to strengthen existing law to facilitate prosecuting offenders. A case in point is trespass.

Under current law, it is illegal for someone to enter someone else's home even if he doesn't take anything or cause any damage. Unauthorized access to a computer, in and of itself, does not constitute a federal crime, however.

Furthermore, the concept of damage is not well defined. For example, has "damage" occurred if the only result of a trespass is a delay in systems operation or time spent to determine that nothing was altered?

The answers to this and other legal questions are complex. Unfortunately, they are being addressed piecemeal by a long list of new bills which have been introduced over the last two years.

All of these are aimed at valid problems. But the Subcommittee did not endorse any of them because we feel that a more global approach is required -- one that transcends the boundaries of a particular agency or Congressional committee.

Looking back, what do I think the Subcommittee accomplished by holding these hearings? Chiefly, I believe we achieved what most Congressional hearings of this kind seek to achieve. We focused public attention on a problem of national importance. And judging by the interest we've received in the Subcommittee's report, I think it is fair to say we were highly successful in doing that.

But what happens from here on? Obviously the issues haven't gone away, nor are the solutions in hand. The break-ins continue. Recently, as many of you know, hackers struck again -- a familiar story. This time they got in through the Southwest Bell System in Kansas City and gained access to computers in major corporations all over the country, fooling the long-distance billing mechanism in the process.

Well, several things are going on right now. One thing is a follow-on day of hearings -- starting at 1:30 this afternoon -- before my Subcommittee. In these, we plan to continue calling the public's attention to the security and privacy implications of our dependence on computers. We also intend to keep urging the Administration to take action.

Another thing which involves me personally, as a Member of the Judiciary Committee, is an amendment I offered to H.R. 5616, the Counterfeit Access Device and Computer Fraud and Abuse Act of 1984. My amendment makes it a crime to gain unauthorized access to financial information and credit reports.

My rationale was that such information is protected by law from release without authorization from the indivudual. So it only makes sense that it should be illegal for a hacker to gain access to it.

H.R. 5616 has so far passed the House; and I am hopeful that it will also pass the Senate and become law next year.

In the longer term, what happens next depends in large measure on you, the experts in this audience. The public is beginning to appreciate the dangers. The Congress is energized. But, it will be up to you to build on this support by developing the detailed answers.

DOD/NBS COMPUTER SECURITY INITIATIVE CONFERENCE

MARVIN SCHAEFER

DOD COMPUTER SECURITY EVALUATION CENTER

Welcome to the seventh jointly-sponsored Conference on the Computer Security Initiatives of the Department of Defense and the National Bureau of Standards.

Since 1979, the year of the first meeting in this series, attendance and active participation have grown with its progression from seminar to symposium to conference. A record of more than 500 persons had preregistered for this Conference by the time these words were being written.

The growth in participation represents a growth in computer security awareness as well as a parallel maturation of the technology and its use. At the time of the first seminar, the community perceived less of a need for computer security than it does today. There had not yet performed their publicized attacks on systems; the popularization of computer piracy on television and in comic strips had not yet brought computer crime to the public mind. Nor was there a Computer Security Evaluation Center with a published Trusted Computer System Evaluation Criteria or a populated Evaluated Products List.

The combined efforts of the National Bureau of Standards and the Department of Defense have brought about the progress that has been made in the last five years. The private sector has begun to produce commercial trusted computer products for subsequent inclusion in their product lines. Such products are being developed, generally without government subsidy, in all of the classes defined in the "Orange Book", which was distributed at our Sixth Conference. In addition, the DoD has begun to incorporate precise interpretations of the Criteria in their procurement specifications for new application systems.

Recognizing the progress that has occured, this Conference represents a significant departure from the agendas of our previous Conferences. We chose to issue a Call for Papers to computer security practitioners and researchers in order to provide a better forum for the exchange of current information about advances in the state of the art as well as the expression of technological challenges for the future.

As in previous Conferences, many of the papers deal with issues relevant to trusted computing bases designed to support military computer applications. It is clear that the nation's security depends on our ability to protect classified information for unauthorized disclosure, modification or destruction. Trusted computing products in Divisions B and A of the Criteria are designed to control accesses between cleared users and classified data. We in the Department of Defense will continue to emphasize the need for developing trusted systems in these Divisions. A primary goal in this Conference series is to disseminate information on trusted computing system technology to facilitate the general improvement of security in commercially-available systems.

In this light, it is interesting to recall that this is the year 1984. Large-scale networks of automated information systems have been realized that outstrip Orwell's visionary prophecy of a technology having the potential to destroy the privacy of the individual. Social Security numbers have become universal identification codes for people living or working in the United States.

Electronic banking, electronic mail, and electronic records-keeping have become a way of life for tens of millions of our population. The national economy has literally been placed under the control of computers and computer programmers.

It is perhaps surprising to observe that by far the largest application of computers in the Department of Defense deals not with classified data, but rather with sensitive but unclassified information. These DoD computers store medical records and data on patients in DoD hospitals; they process payroll and retirement funds; control the acquisition, shipment and disposal of equipment; arrange the transportation of personnel and material on airplanes; etc. Such DoD applications are no different from the majority of computer applications in the bulk of the Federal government and in the private sector.

The computer security threats to these computer systems are no different from those to computer applications in most of the Federal government and in the private sector. The audit and control requirements for these applications are no different from those for the bulk of applications in the Federal government and in the private sector. The traditional control requirement is for the establishment and maintenance of protected audit trails that contain a record of each "security relevant" system event, identifying the individual responsible and accountable for each of these events along with the context of the individual's actions.

Such audit trails can quickly become a large, sensitive database. Specialized tools need to be developed to process and analyze such audit trails in order to help discover evidence of computer fraud or abuse. However, this is where an Orwellian challenge comes to mind. The very records required by the auditors can become a threat to individual privacy. It is almost as though individuals need to sacrifice their privacy in order to protect their privacy!

Several papers in the Conference Proceedings address these concerns.

Advances in computer security research and technology are addressed in many of the Conference sessions. Heavy emphasis is given to recent directions in database management security, network security, and applications of interest to the private sector. We have attempted to schedule sessions such that participants can be exposed to reports and discussions of recent progress, to planned thrusts in computer security, and to current problems that require solution. That computer security is now in practice should be evident to the participant and to the general reader of these Proceedings. We look forward to your support in continuing to advance the technology base.

This Conference resulted from the prolonged efforts of a small group of dedicated, but relatively invisible, professionals. I would like to acknowledge the work of the Program Committee; Daivd Bell and Sheila Brand from the Center, and Denny Branstad and Stu Katzke from NBS have done a splendid job of recruiting, reviewing, and then selecting the papers and presentations that comprise this Conference's sessions. The difficult and sometimes thankless task of planning, coordinating, and organizing the Conference's mechanics and publications was done by Al Duke, Carolyn Logan, Mimi Vaughn, Tammy Shelton, and Carol Atkinson of the Center; and Gene Troy, Kathy Kilmer, Sarah Torrance and Jan Kosko of NBS. Finally, more than sixty individual computer security practitioners and researchers in the United States and abroad submitted professional papers for inclusion in this year's program. Despite short notice and publication deadlines, we received tremendous support from the computer security community. On behalf of the Program Committee, I would like to express my appreciation for all their contributions.

COMPUTER SECURITY IN PRACTICE

JOSEPH S. GREENE, JR. COLONEL, USAF

DEPUTY DIRECTOR DOD COMPUTER SECURITY EVALUATION CENTER

On the 2nd of July, I joined the Computer Security Evaluation Center as the Deputy Director. Although computer software engineering has been a common thread through my prior Air Force assignments, the subject of trusted computer base technology represents a new challenge for me. For this reason, I am particularly pleased with the opportunity this conference provides to meet those of you who have a head start on security.

The significant event proceeding last year's DoD Computer Security Initiative Conference was the publication of the Department of Defense Trusted Computer Systems Evaluation Criteria in August of 1983. Actually on the first day of the Conference -- that was 15 November 1983 -- the Under Secretary of Defense for Policy published a Memorandum to the Defense Components that authorized and encouraged use of the Criteria in the performance of security evaluations and as the basis for system security requirements. The first question we could asked is what has happened during the past year since the Criteria was published.

In the area of Product Evaluation, I am pleased to report that the Evaluated Products List (EPL) is no longer an empty set. Two reports have been published. The IBM Resource Access Control Facility called RACF has been given an overall evaluation class of C1. RACF is designed to limit access to resources by identifying authorized users and protected resources and controlling users' access to those resources. It appears that RACF could be evolved to meet the C2 level of evaluation criteria with the addition of the object reuse capability.

The second entry on the Evaluated Products List is the Access Control Facility II built by SKK, Inc. ACF2 was determined to meet all of the requirements of class C2 with no exceptions. ACF2's strong discretionary access controls and audit features provide significant improvements to the security of the IBM MVS operating systems.

The draft final report on CGA's Top Secret add-on security package assigns a C1 rating. The package will allow users to protect project or private information by keeping other users from accidentially reading or destroying that data. Evaluation of Honeywell's Security Communications Processor, called SCOMP, has been completed and with the correction of a few items (primarily in the documentation area) is expected to be placed on the EPL at the A1 level before the end of the year.

During the past year, the Criteria has also been used for the development of future military systems security requirements specifications. Procurement specifications for the BLACKER and the I-S/A AMPE have been prepared based on the Criteria and call for class A1 products. The Center recommends these as good examples of security procurement specifications. A bad use of the Criteria combines features from different class levels defining, as it were, a new evaluation class. We are concerned that this approach, if allowed to continue, could undermine the DoD strategy of populating an Evaluated Products List by sending mixed signals to industry as to our requirements.

In order to discourage this practice, the Director of NSA has recommended to the Secretary of Defense that program managers be directed to explicitly state computer security requirements in terms of a specific objective protection class as defined in the DoD Trusted System Evaluation Criteria. This recommendation is currently being reviewed at the OSD.

During the past year, we have develop an environment doctrine guide to help system managers analyze their mission and environment to determine prudent levels of security for their area. The document, to be described in detail by the next speaker, is being coordinated now.

The Center has used the Criteria and the environment doctrine as the basis for a standard system evaluation questionnaire. The questionnaire has been used to conduct a survey to assess the security posture of automated information systems handling classified and other sensitive DoD information, within the department. A preliminary analysis of approximately 7000 survey responsives covering about 15000 computer systems has interesting results. About one third of these systems process some form of classified data. Of these classified machines, 62% of the systems reviewed are operating outside the region of acceptable risk as defined in the environment doctrine guidance and 17% of the systems review are operating today at a level of trust outside of available technology. That situation is certainly not good.

We urgently need to get on with the task of developing technologies to bring current and projected operations within prudent levels of security risk. Although we have made significant progress since the Center was established in January 1981, it is our assessment that the development of the improved security measures is not keeping pace with the proliferation of internetted and networked information systems. In fact our exposure to unauthorized access is probably greater today than it was four years ago for two reasons: (1) security technology is not significantly more available and (2) the number of computers accessible remotely has greatly increased. This very real and growing gap between operational security need and the existing technology base is one measure of urgency for a strong research program.

One reason for the slow pace of evaluated product development and, hence the widening gap derives from the methodology and processes used to certify new products and systems. Today's approaches are fundamentally human-expert intensive, manual efforts. Although the Center has initiated an education and training program, we are not training new people in sufficient numbers to keep up with the demand for new system evaluations. Once assigned to a product or system evaluation task, people may be largely unavailable for new work assignments for several years. The Center has been forced to ask the services and agencies to prioritize their security evaluation needs and only the most important are undertaken.

Looking ahead, we realize that we are on the bowwave of a very large endeavor. There is no reason to expect that once evaluated and certified, systems and products will remain unmodified. And yet good security practice requires that modified products must be reevaluated. The second and subsequent recertification efforts would be expected to go faster if the original team members are still available, but that may not always be possible.

As a community we have not come to grips with the fact that the availability of trained, trusted-systems designers, specifiers and developers as well as evaluators is a limiting factor. The situation can only become more difficult if

we do not find new ways of doing business. Although the data are insufficent to project accurate work load trends, we are confident that the human wave approach can not keep up with demand. We must find automated supplements and replacements for the present people-intensive evaluation approaches. The situation is going to get worse before it gets better. We must immediately add and train more people to handled the immediate demand for additional evaluations. Simultaneously we must greatly increase and focus our research initatives on developing and promulgated trusted computer systems design and implementation approaches that are conducive to an evaluation process requiring reduced levels of manual effort. System designs that do not address computer security requirements increase the level of manual effort for evaluation and decrease the level of confidence in the resulting evaluation. Research effort on formal specification, verification and eventually automated theorem provers must be guided by the goals of acquiring and making available tools that reduce the personnel burden, as well as providing a high degree of assurance in computer security. Such tools must be developed and made widely available to industry so that more of the burden of certification can be placed upon the system developer and contractor.

Because there are only a handful of trusted software experts in the country, and even fewer trusted hardware experts, the Computer Security Evaluation Center must adopt approaches for leveraging the maximum return from the people we have while we educate more people. In the remainder of my talk, I want to focus on the leverage opportunities we have identified.

I want you to understand what we are trying to do and why so that you can help us achieve our objectives. The approach I'm going to discuss, while completely consistent with the philosphy of partnership with industry on which the center was chartered, adds a new dimension to the existing approaches that will be continued. As you examine our ideas, keep in mind that our fundamental objective is to cause more products to be available sooner. So far, the Center has attempted to motivate industry to carry the burden of security development through a indirect route of establishing standards, evaluating products, and providing access to formal specification and verification tools.

This passive approach gives us no control over capability or schedule. We are left largely at the choice of industry as influenced by the market-place to move ahead with trusted products. This passive approach is not leading to the development of trusted products at a rate sufficent to support DoD requirements.

While continuing to motivate industry through the indirect approach, the Center will additionally accelerate the process by sponsoring development of exemplary trusted system prototypes. In this undertaking, every effort will be made to use software engineering practices that facilitate development of machine independent products that offer maximum reusability. We will focus effort on the development of common packages of logic that can be used by many vendors. With this approach the Government will encourage and foster wider competition and can therefore more easy justify the associated development cost. We plan to place Government sponsored exemplary implementations in the public domain with wide availability to vendors. Of course some parts of the code must deal with machine dependencies. We will seek to isolate and carefully document these portions in the expectation that vendors would undertake the task of developing the special-machine dependent packages that interface the common software to their particular hardware base. Under this stratgey, industry should be able to design, develop, and market trusted product with less risk than in the past, since they will be able to obtain clarification on how well the product is likely to satisfy the requirements of the criteria and they will be able to draw directly from the

common work sponsored by the DoD. The approach promises cost and schedule leverage opportunities for both government and industry.

Reusable machine independent software requires a tightly controlled language standard. The DoD Ada computer language standard and the Ada compiler validation facility make Ada the most tightly controlled standard available today. Additionally, the DoD has made a major commitment to the Ada computer language. This action alone will naturally give preferential treatment to those vendors that supply future technology base for our primary and most important DoD mission-critical customers.

We have decided to undertake as an additional research thrust, a commitment to develop trusted technologies in and for Ada in order to obtain the maximum leverage within the department. As you know, some within the security community have raised concern about the feasibility of using Ada to implement trusted systems. With our commitment to Ada, the Center is taking on the task of resolving these questions for the community. In order to move out in this new direction, we have already undertaken a number of actions. We will use 15 of the FY-85 manpower space increases to hire new people with Ada experience. These people will develop Ada foundation technologies like trusted operating systems technologies, trusted data management tools, and common telecommunications protocol in and for Ada. We will also began to develop and make available tools for validation and formal verification in Ada. We are in the process of redirecting some of the consolidated computer support program activities toward implementations in the Ada language. If requested additional FY-85 reprogrammed funding is approved, we will increase our contractor support in this new direction.

During the approximately 24 months prior to my assignment to the Computer Security Center, I helped evaluate over 330 proposals from industry leading to the award approximately 50 contracts totalling over 15 million dollars. Much of that work involved development of software in Ada. One of the lessons that I learned from that effort was the critical importance of contracting mechanisms to facilitate the rapid advancement of technology. We are working with our contracting officers to develop the infrastructure to facilitate a responsive contract award process. We will move out as soon as we can establish these mechanisms and obtain needed funding. We are exploring the feasibility of sponsoring briefings for industry on our research objectives. We hope to use these mechanisms to establish an incentive for creative and new thinking on our problems.

We believe also recognized that there is considerable leverage opportunity for the Center by working with standards committees, by taking a more activity role in the Government review of industry sponsored IR&D security related efforts, and by encouraging a more active and formal exchange of information on security work sponsored by other Government organizations. We are establishing points of contact and mechanisms to facilitate our involvement in these areas.

In order to further strengthen our partnership with industry, we are exploring ways to protect the market incentive for industry to invest private sector dollars in security related software products. Today, proprietary software is generally distributed in machine code form to reduce the opportunity for piracy. This practice also increases the potential for subversion. The Computer Security Center will undertake research to develop ways to accept proprietary software that reduce the risk of subversion.

This may require that software be delivered in source code form, be examined by the Government, compiled on Government machine and distributed under Government control in machine code form. To be practical the approach cannot void vendor warranties and must be responsive to a reasonable updating process. We believe the DoD move to Ada will facilitate this goal. If successful, this research could lead to the establishment of a trusted software acceptance facility for the department.

As we look at the life-cycle systems implication of security, we recognize that a related capability needs to be developed to place hardware configurations deemed applicable to the DoD's most sensitive processing applications under strict configuration control similar to those used to protect communication security hardware from subversion. We plan to work closely with vendors to develop the appropriate controlled-access hardware development environments within industry as a cost effective alternative to forming a totally Government control hardware verification facility for each vendor's product. Random sampling techniques in which the hardware is periodically subjected to scrutiny in a Government laboratory should serve to provide assurances that the hardware continues to meet security specifications.

Our recommendations calling for three new initiatives dealing with exemplary software development, a software acceptance facility, and a hardware verification capability, represent new directions for the Computer Security Center. These efforts should accelerate the availability of trusted products and must be undertaken in a way that foster a growing partnership with industry.

These and other recommendations for greatly improving security measures and reducing the growing exposure to unauthorized computer access are being examined within the department.

I hope you will take the opportunity during the conference to introduce yourselves. I will look forward to getting your view on the ideas that I have proposed. We are looking for new ways to get on with our common objectives of increasing the availability of trusted software products.

Environmental Guidelines For Using the DoD Trusted Computer System Evaluation Criteria

by

Sheila L. Brand

DoD Computer Security Center

1.0 Introduction

In August of 1983 the Department of Defense Computer Security Center (DoDCSC) published the document entitled: Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83.(1) The evaluation criteria defined in that document classify systems into seven hierarchical classes of enhanced security protection. They provide a basis for the evaluation of effectiveness of security controls built into automatic data processing system products. The criteria were developed for the following reasons:

* to provide users with a yardstick with which to assess the degree of trust that can be placed in computer systems for the secure processing of classified or other sensitive information

* to provide guidance to manufacturers as to what to build into their new, widely-available trusted commercial products to satisfy trust requirements for sensitive applications

* to provide a basis for specifying security requirements in acquisition specifications

Two types of requirements are described in CSC-STD-001-83. They are: (1) specific security feature requirements, and (2) assurance requirements. The underlying assurance requirements can be applied across the entire spectrum of Automatic Data Processing (ADP) system or application processing environments without special interpretation. However, though the criteria are application independent, it is recognized that the specific security feature requirements may have to be interpreted when applying the criteria to specific applications or other special processing environments.

Though CSC-STD-001-83 provides the yardstick with which to measure the degree of trust, it does not provide guidance on how much trust is needed. That is, it does not provide a mapping of the criteria classes to protection requirements of processing environments of varying degrees of risk. In order to fill this void and thereby provide guidance to DoD system managers and others responsible for designing or procuring secure systems or evaluating the effectiveness of controls in operational systems, the DoDCSC has formulated doctrinal guidance for using the DoD Trusted Computer System Evaluation Criteria.

2.0 Scope

The doctrinal guidance has been developed with the objective of being applicable to DoD systems that are entrusted with the protection of a wide range of information. It addresses national security related classified and unclassified information. It also covers sensitive and nonsensitive unclassified information that is not national security related.

3.0 Determination of Risk

In order to determine the minimum criteria class necessary to protect information processed by a system it is first necessary to determine the security risk inherent in that system. Looking for parameters that would be universal to all DoD systems, the DoDCSC chose two which are commonly used throughout the Defense community. They are:

* sensitivity/classification of information processed by the system - which is a measure of the value that DoD places on information

* clearance of system users - which is a measure of trust DoD places in users

3.1 Risk Index

Using the parameters of sensitivity and clearance, the DoDCSC formulated that the inherent risk in a system, to be designated as the RISK INDEX of a system, is defined as the disparity between the maximum clearance or authorization of the least cleared system user and the maximum sensitivity of data processed by a system. In other words, RISK INDEX is defined by the disparity between data sensitivity and user trust.

In order to calculate the RISK INDEX for a system it is necessary to assign numeric rating values to the range of sensitivities and the range of clearances.

Using assigned numeric ratings the risk index can be calculated:

RISK INDEX = Rmax - Rmin

where:

Rmax = rating associated with the system's maximum data sensitivity

Rmin = rating associated with the maximum clearance of the least cleared system user

The rules for arriving at Rmax, the rating for maximum information sensitivity, also take into account the presence of non-hierarchical sensitivity categories such as NOFORN (Not Releasable to Foreign National) and PROPIN (Caution - Proprietary Information Involved). The term "sensitivity categories" also encompasses compartmented information and information revealing sensitive intelligence sources and methods.

4.0 Open and Closed Environments

In addition to user clearance and data sensitivity the DoDCSC formulated that two types of environments had to be accounted for in arriving at the appropriate Criteria Class. These deal with the environment in which the application is developed and maintained.

A system whose applications are not adequately protected is referred to as being in an open environment. If the applications are adequately protected, the system is said to be in a closed environment.

Here application refers to those portions of a system including portions of the operating system that are not responsible for enforcing the security policy.

4.1 Closed Environment

A closed environment is defined as one in which both of the following hold:

* Application developers (including maintainers) have sufficient clearances and authorizations to provide acceptable presumption that they have not introduced malicious logic. Sufficient clearance is defined as follows: where the maximum classification of the data to be processed is Confidential or less, developers are cleared and authorized to the same level as the most sensitive data; where the maximum classification of the data to be processed is Secret or above, developers have at least a Secret clearance.

* Configuration control provides sufficient assurance that applications are protected against the introduction of malicious logic prior to and during the operation of system applications.

4.2 Open Environment

An open environment is defined as one in which either of the following holds true:

* Application developers (including maintainers) do not have sufficient clearance (or authorization) to provide an acceptable presumption that they have not introduced malicious logic.

* Configuration control does not provide sufficient assurance that applications are protected against the introduction of malicious logic prior to and during the operation of system applications.

The objective of differentiating between open and closed environments, in terms of risk, is to take into account the possibility of the insertion of malicious logic during the system's development and maintenance phases. The presumption is that systems in open environments are more likely to have embedded malicious software than those developed and/or maintained in a closed environment. Today, most systems are developed and maintained in an open environment.

5.0 System Users

In making the determination of RISK INDEX the analysis must address two types of possible users. These are:

* direct users: users with direct access to the system; that is, users who can provide input to or obtain output from the system without the intervention of another human, and

* indirect users: users who do not have direct access to the system, but who can still provide input or obtain output from the system.

While it may be obvious why direct users must be accounted for in the determination of RISK INDEX, the role of indirect users may not be as obvious. To understand their importance consider the following example: Suppose that there is a system operating in the System High Mode at the Secret level and this system automatically labels its output. By extension, all direct users are cleared and are trusted to at least the Secret level.

Under current DoD policy, all output produced by this system should, before release, be manually reviewed and assigned the proper sensitivity markings by a responsible authority. If this is done, then there are no indirect users, and the Rmin value used in calculating the RISK INDEX is that associated with the least cleared direct user.

However, if there is reason to believe that the output will not be manually reviewed prior to release, the element of trust that was previously placed in the responsible authority reviewing the output is now placed in the computer system itself. The labels that this system places on its output are trusted to be accurate, and the output is distributed according to the labels. In the example, if a listing is marked Confidential by the system it may be sent to someone with only a Confidential clearance. This person should then be considered an indirect user, for he has received output from the system, and the distribution of that output was based solely on the label attached by the system.

At this point, the system security officers should carefully examine the circumstances to determine whether or not they are truly operating in System High Mode. The assumption was made that all system users are cleared to at least the Secret level, and yet someone with only a Confidential clearance has just received output from the system without it being manually reviewed. The system, therefore, is being trusted to accurately separate and label various sensitivity levels of data (at least Confidential and Secret, in this example). This assumption of trust is not required of systems operating in System High Mode.

As the example shows, failure to take into account the possibility of indirect users may result in an underestimate of security requirements. In this case, calculation of RISK INDEX, using the direct user's clearance for Rmin, would result in requirements for a system possessing need-to-know protection. However, when the indirect user's clearance is used for Rmin the resulting RISK INDEX indicates a need for mandatory access control protection.

6.0 Doctrinal Guidance

As of the writing of this paper The DoDCSC had not yet finalized the document containing the doctrinal guidance. However TABLE 1 is provided in order to give the reader a draft representation of the guidance.

TABLE 1: Security Index Matrix For Open Security Environments, illustrates the results of applying the doctrinal guidance to individual minimum clearance/maximum data sensitivity pairings, where no categories are associated with maximum sensitivity below Top Secret.

7.0 Conclusion

The DoDCSC has formulated doctrinal guidance to be used with The DoD Trusted Computer System Evaluation Criteria by identifying the minimum Criteria Class of system required for a given RISK INDEX. RISK INDEX is defined as the disparity between the minimum clearance or authorization of system users and the maximum sensitivity of data processed by the system.

The purpose for developing this guidance was to make it available for use in establishing minimum computer security requirements for the processing and/or storage and retrieval of sensitivity or classified information by the DoD whenever automatic data processing systems are employed.

REFERENCE

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

TABLE 1

SECURITY **INDEX** MATRIX
FOR OPEN SECURITY ENVIRONMENTS[1]

Maximum Sensitivity of Data

Minimum Clearance or Authorization of System Users	U	N	C	S	TS	1C	MC
U	C1	B1	B2	B3	*	*	*
N	C1	C2	B2	B2	A1	*	*
C	C1	C2	C2	B1	B3	A1	*
S	C1	C2	C2	C2	B2	B3	A1
TS(BI)	C1	C2	C2	C2	C2	B2	B3
TS(SBI)	C1	C2	C2	C2	C2	B1	B2
1C	C1	C2	C2	C2	C2	C2[2]	B1[3]
MC	C1	C2	C2	C2	C2	C2[2]	C2[2]

[1]Environments defined in the shaded area are for systems that operate in system high mode. No minimum level of trust is prescribed for systems that operate in dedicated mode. Categories are ignored in the matrix, except for the inclusion of compartments at the TS level.

[2]It is assumed that all users are authorized access to all compartments on the system. If some users are not authorized for all compartments, then a class B1 system or higher is required.

[3]Where there are more than two compartments, at least a class B2 system is required.

U = Uncleared or Unclassified
N = Not Cleared but Authorized Access to Sensitive Unclassified Information or Not Classified but Sensitive
C = Confidential
S = Secret
TS(BI) = Top Secret (Background Investigation)
TS(SBI) = Top Secret (Special Background Investigation)
1C = One Compartment
MC = Multiple Compartments

WORKING TOWARDS A1

D. Elliott Bell

Department of Defense
Computer Security Center

Abstract. Building a system to meet the A1 requirements for certification necessitates scheduling work activities and documentation related to A1 certification within a total system development schedule. This paper proposes one approach to synthesizing development activities and A1 certification requirements.

INTRODUCTION

With the advent of the "Department of Defense Trusted Computer System Evaluation Criteria" CSC-STD-001-83 (hereafter called the "Criteria"), system developers and system acquirers were put in the position of being able to specify very precisely degrees of computer security. Initial consideration, however, identified a gap between the A1 requirements in the Criteria and the process of system development. From a development point of view, it is as if the Criteria say "At the end of the development process, the collective engineering notebooks will contain the following." The Criteria do not (and probably should not) detail how the activities and their documentation should be monitored and reviewed, nor even how they should be ordered. This paper will describe one approach to organizing A1 certification activities and documentation within the framework of a typical system development schema. First, both the A1 requirements and the development process will be summarized. Then a synthesis that grew out of consideration of two actual programs - BLACKER Phase 1 and the Inter-Service/Agency Automated Message Processing Exchange (I-S/A AMPE) - will then be presented along with some unresolved considerations.

A1 REQUIREMENTS SUMMARY

The Criteria divide requirements into feature requirements and assurance requirements. For the purposes of this paper, the A1 requirements will be viewed as quadripartite: system features; assurance formalisms; activities; and documentation.

The system features that are required concern security policy, accountability, system integrity, and trusted facility management. Also included are system architectural features such as domain separation, process isolation, the object abstraction, segmentation, and layering that have historically been shown to simplify assurance. One feature (audit) is present to the extent that it is at least in part to offset a known area of vulnerability, that of covert channels. Thus the full list of auditable events is tied to the successful activity of covert channel analysis and its documentation. The full set of features required is an amalgam of direct security transliterations and supplementing preventive and monitoring techniques.

The assurance formalisms include secure system architecture, minimalization of the Trusted Computing Base (TCB), the principle of least privilege, an implementation of the reference monitor, and a full design "certification chain". The design certification chain begins at the security policy and proceeds through the security policy model to both the Detailed Top-Level Specification (DTLS) and the Formal Top-Level Specification (FTLS) and finally to the coded implementation.

The activities required for A1 certification are of several types. There is a set of consistency and correspondence activities: show that the FTLS accurately represents the TCB; show that the FTLS is consistent with the security policy model; and so on. There are also testing and analysis. Analysis of the specifications for covert channels must be undertaken, results quantified, and, as directed, channels reduced or eliminated. In addition, an implemented system must undergo full-scale security testing to discover design and implementation flaws. Discovered flaws must be rectified. Finally, some support activities are required, notably configuration management (both of the usual design documentation and of the special A1 documentation) and trusted distribution.

Documentation to support an A1 certification runs a wide gamut from definition and justification of the design certification chain to a security users' guide. In general, every feature, every assurance formalism, and most of the activities are required to be documented.

SYSTEM DEVELOPMENT PROCESS

The system development process, encompassing both hardware and software development, is divided into phases normally delimited by formal review of program documentation. For the purposes of this paper, the paradigm shown in Figure 1 will be adopted. There are five phases - the definition of requirements, design, build, test, and acceptance. The phases overlap, the overlaps highlighted by shared documentation and a program review. The requirements definition phase ends with the System Requirements Review that addresses the adequacy of the Functional Description of the system to satisfy the needs documented in the Statement of Requirements. The design phase begins at the System Requirements Review and consists of a three-step refinement of the Functional Description. The three evolving design documents are the System Specification (reviewed at the System Design Review), the System/Subsystem Specification (reviewed at the Preliminary Design Review), and the Implementation Specification (reviewed at the Critical Design Review). At the Critical Design Review, the development process enters the build phase. The Implementation Specification is the basis for the building of actual modules, whether the process is the writing of computer programs or the fabrication of hardware items. The build phase ends when the modules have been tested individually and the test phase begins. (The demarcation between the build and test phases is not as clear as that between the earlier phases inasmuch as the testing interacts with the building and the progress on different subsystems proceeds at different rates.) Testing proceeds from modules towards the full system, culminating with a test of the total system during the Developmental Test & Evaluation. This test activity and its documentation initiates the acceptance phase which includes both an Operational Test & Evaluation and a final acceptance test and a sign-off document.

This representative system development schema can illustrate several philosophical points about the system development process. First, the underlying assumption is that there is a distinction between the system developer and the system acquirer or "user" who promulgates the Statement of Requirements. Even if a system is developed within a single organization, this assumption tends to be supportable as the developer and user roles are adopted by different subelements of the organization. Second, the review milestones within the development process supports a compromise between the developer and the user. The user needs to be assured periodically that the development is progressing well. The developer needs some assurance that the user understands and approves the direction that the development is taking. The development reviews and the documents that support them (particularly the design reviews) are incremental program definitions where the developer and user jointly agree to proceed and to share the risk and the potential cost of good faith errors that may be found later in the process. All the development activities and documentation are fit into the same incremental progress schedule. For example, logistics planning and test plans are undertaken in parallel with the design activities and the relevant documents and milestones are scheduled so as to fit naturally into the larger development review schedule.

The Al computer security certification requirements can be viewed as an additional set of requirements that need to be added to the development schema. The question is "Where do the Al requirements fit in the larger system development context?"

PROPOSED SYNTHESIS

Inserting the Al computer security requirements into the system development process is not nearly as formidable as it seems at first blush. Most of the requirements fold in neatly as minor additions and extensions to usual design and documentation tasks. The primary exceptions are the additions of a design certification chain and of covert channel analysis.

System Features. The system features required for Al certification constitute a checklist of items that should be reflected in the usual design and user documentation. This checklist is of use to both the user and the developer and should, in fact, act as a memorandum of understanding concerning the system features that will be scrutinized carefully for computer security relevance. Three particular features deserve more comment. (1) The access control features are inextricably tied up with the assurance formalisms of security policy and its enforcement. The review of these features must of necessity be coupled with a review of the corresponding elements of the design certification chain. (2) The user functionality features of trusted facility management, while directly addressing security issues, really embody just one more example of an identified class of user. The requirement to include such features and to document them in the trusted facility manual is not, therefore, a new type of requirement, but rather another application of a standard system development task. (3) The audit function for a system working towards Al certification is important, extensive, and incompletely defined at the outset of system development. The full definition of the audit function itself and of the list of auditable events is arrived at through several of the activities required in an Al system development, particularly covert channel analysis and routine design analysis.

Assurance Formalisms. With one major exception, the assurance formalisms are supported by computer-security-related activities and are reported in

specially tailored design documentation. Such items as the secure architecture, the definition and justification of the TCB perimeter, the application of least privilege, and the instantiation of the reference monitor concept fall into this category. The design certification chain is more complicated. Two elements of this chain, the DTLS and the software code, are not new requirements. The other elements are new, as are the links between them. An important point is that the DTLS consists of a set of design documents, each one a refinement of the one before. Experience in system development has shown that the compromise of publishing, reviewing, and approving design documentation at several milestones (design reviews) leads to a better appreciation by both the user and the developer of the direction and the progress of the effort. In an analogous fashion, the FTLS is usually not monolithic but consists of levels of refinement just like the DTLS. The Criteria call for an FTLS that corresponds to the model, that is internally consistent, and that corresponds to the implementation. A developer might well prefer to present the FTLS and its assurances late in the development schedule. That approach would, however, give the user no ability to gauge the progress of the development of the FTLS. As a compromise, therefore, it is proposed that the refinement of the DTLS and the FTLS be kept in rough lockstep. That is, that at the major design reviews a version of the FTLS comparable in level of detail to the design documents under review be presented, reviewed and approved. Exactly equivalent risk will be jointly assumed by the user and the developer, as in the case of the DTLS. Additionally, by assuring that the developing DTLS and FTLS describe the same design, confidence in the consistency of each set of refinements can act synergistically to enhance the total confidence in the design. Moreover, the requirements to show that both the DTLS and FTLS correspond to the model and to the implementation code should be able to interact to mutual benefit.

Activities. Five major new activities must be added to the system development schedule. (1) The development of the design certification chain (particularly the FTLS) should proceed in lockstep with the development of the DTLS, as described above. (2) The covert channel analysis should be undertaken after the FTLS and DTLS are available, namely after successful completion of a Critical Design Review. It is true that any necessary revisions uncovered during the build or test phases might necessitate repeating this analysis, but since the probability of a major revision being necessary is low, the risk is worth taking. (3) Security testing to discover security protection flaws should be undertaken after a stable system exists. This testing should therefore be done in parallel either with the Developmental Test & Evaluation or with the Operational Test & Evaluation. The decision is subjective and is based on the potential for design change during testing. An important decision that must be made early in a development is what role the developer will play in the security testing. "No role" is probably not reasonable. The spectrum runs from training the security test team through providing back-up consultant support to providing team members. The impacts on the "contract" are substantial and the issue should be resolved as soon as is feasible. (4) A configuration management program for both normal system development material and for AI-specific material must be executed beginning at the start of the design phase and lasting until system acceptance. (5) The trusted distribution activity should begin late in the test phase and is a direct extension of the configuration management program.

Documentation. The documentation required for AI certification falls into two classes: information tht fits naturally into normal system development documentation and information that is idiosyncratic to AI certification.

Figure 2 illustrates one possible packaging for A1 documentation. Some of the new documents (particularly the Design Certification Document) combine several A1 requirements. Since the component topics may be addressed at different times, a prudent approach to making documentation available when needed while minimizing unnecessary cost and effort rewriting documents is clearly called for.

SUMMARY

Adding computer security requirements to the system development process when working towards A1 need not be difficult. The security features form a checklist to be used in the development process. The assurance formalisms consist of design analyses and the creation of a design certification chain. The development of the FTLS and the DTLS should be intertwined, kept in rough lockstep, and reviewed in parallel during the design reviews. Most of the required activities fit naturally into the development schedule using standard design analysis documentation formats. The covert channel analysis has no traditional analogue, but should be initiated after the Critical Design Review. Based on the role chosen for the developer in security testing, straightforward planning will add routine tasks and documents to the schedule. The documentation requirements are satisfied either by tailoring standard system development documents or by adding A1-specific documents to the list of deliverables.

This proposal covers all the requirements for A1 certification, but it leaves several issues unresolved. One that has been mentioned is the role of the system developer in security testing. Another is the exact manner of intertwining the FTLS and DTLS development: should the DTLS lead the FTLS or the reverse? The decision is immaterial to the user but can have a marked effect on the developer. Managing the effects of activities on features (covert channel analysis on audit) or of assurance formalisms on features (security architecture on access control features) is fraught with the possibility for failure. Clearly this plan can aid the potential developer of an A1 system, but it is by no means the last word about working towards A1.

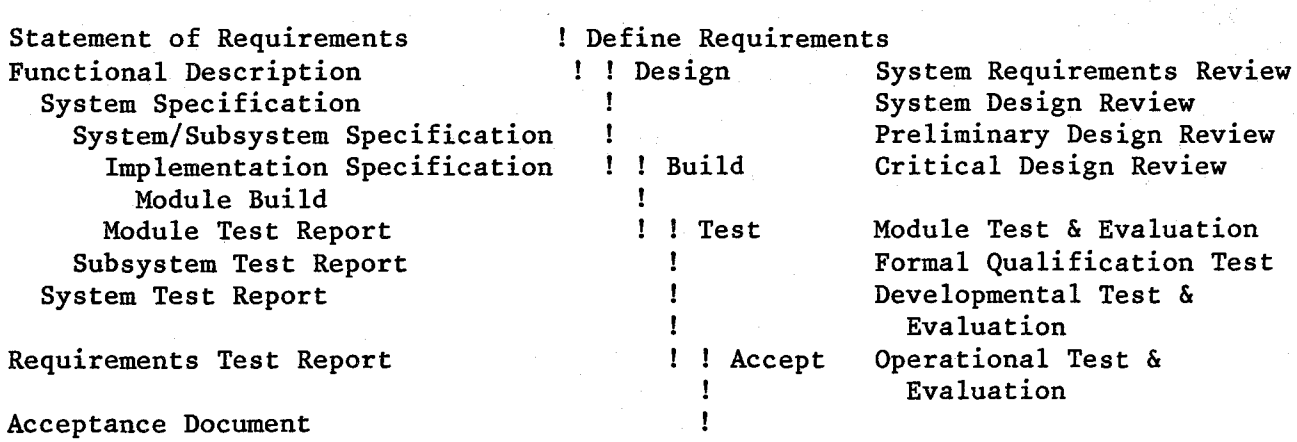


Figure 1. System Development

TAILORED DOCUMENTS	
System Specification	DTLS
System/Subsystem Specification	DTLS
Implementation Specification	DTLS
Users' Manual	Security Users' Manual
	Trusted Facility Manual
	Security Features Guide
Configuration Management Plan	Security Configuration Management Plan
Design Analysis Report	Design Analysis
Test Plans, Procedures, Reports	Security Test Plans, Procedures, Reports
Code	Code
NEW DOCUMENTS	
Design Certification Document	Security Policy
	Formal Security Policy Model
	Model Supports Policy
	Model Internally Sound
	FTLS
	DTLS Cross-Reference
	Model-to-FTLS
	DTLS-to-Model
	FTLS Completeness Proof
	FTLS-to-Code
Formal Verfication Document	Verification Plan
	Verification Tools
Covert Channel Analysis Document	Covert Channel Analysis
System Security Architecture Document	Security Features
(This document could be included	Reference Monitor
in the DTLS documents.)	Process Isolation
	Al System Architecture

Figure 2. Packaging of Al Documentation

Acknowledgements. My thoughts on organizing Al computer security requirements within a system development were formed during a variety of discussions with colleagues. Within the the Department of Defense Computer Security Center, the contributions of M. Schaefer, J. Houser, T. Losonsky, and L. O'Dell were particularly noteworthy. Discussions with C. Savant, G. Cole, J. Hemenway, and D. Cooper at SDC also proved very valuable.

THE PRACTICAL ASPECTS OF MULTILEVEL SECURITY

Terry S. Arnold
Vice President, Technology
Merdan Group, Inc.
4617 Ruffner Street
San Diego, CA 92111

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. 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 issue has been widely discussed over the past 15 years. The positions taken by various people range from that "it is possible" to "nothing less is acceptable." This paper presents the views of one practitioner who believes that it is currently feasible, as long as appropriate constraints are applied by management. The emphasis of the 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 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 even now 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 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 the Department of Defense (DoD) security policy. The Bell-La Padula 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 widespread 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. 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 verifiation of software implementations. Some research groups are working in the area, but solid results may be several years away. 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 of 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 commercial attempt to produce a multilevel secure operating system. This effort is particularly notable, since it has been submitted to the DoD Computer Security Center for evaluation at the A1 (i.e., highest) level. The jury is still out on SCOMP, but 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 is not a failing of software engineering technology, but 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 defines 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 does 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.

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

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 that a given system actually meets is one of the elements of the charter of the DoDCSC. Unfortunately no 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 several vendors have submitted products for evaluation. The impact time 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 Program Manager 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 five years. Even when these products finally are available, a C3I Program Manager 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 gave 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. Figure 1 illustrates the relationship between these regulations and the Executive Order which forms the base requirement. 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 be 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.

Rules as the Basis of Access Control in Database Management Systems

David A. Bonyun

I.P. Sharp Associates Limited
600-265 Carling Avenue
OTTAWA, Canada K1S 2E1

INTRODUCTION

From almost the beginning of the work aimed at the creation of provably secure computer systems, the application area of databases and their management has been of interest and near the top of the list of significant peripheral issues to be pursued. While the primary effort was aimed at operating systems and the reference monitors which they were to contain, other activity was planned and executed in the database area. Both Systems Development Corporation [Hinke 75] and I.P. Sharp Associates [Grohn 76] were contracted to investigate the DBMS problem and both chose to limit their research to those databases which were relationally organized. A secure RDBMS was the elusive goal.

The 1982 summer study session, at Woods Hole, Massachusetts, sponsored by the Air Force Studies Board of the National Academy of Sciences, reviewed the area and determined that the relational model was the most attractive from a theoretical (and hence, it was hoped, from a provable) point of view. Three different classes of database were considered at Woods Hole: the first consisted of data of just two different security levels; the second consisted of data which was largely textual - e.g. messages or documents; the third constituted the general case. In each case, DBMS problems requiring solutions in the foreseeable future were to be addressed as fully as possible given the present state-of-the-art.

These three cases were chosen principally because it was believed that they offered problems potentially solvable in the near, middle, and long terms respectively. Moreover, each case was an abstraction of one or more real situations presented to the session's steering committee by a variety of agencies of the U.S. government.

There was at least one significant common element to emerge from the briefings prior to the Woods Hole session. The agencies who provided the briefings almost all felt that their data ought to be marked at the data element level. Neither the SDC work nor the IPSA work had seriously contemplated that security ought to be enforced at this fine level of granularity. Indeed the IPSA team had opted for attributes to apply to full relations while SDC chose to assign attributes to columns (or domains).

The second of the three cases presents a significantly different picture of the database from the first and third. Databases appear to be separable into two classes depending upon the degree of volatility which the data structures exhibit. Whereas the first and third cases, the so-called "binary" and "general" cases, appear to focus primarily on databases in which the structures are quite rigid, with variable data within them, the second, textual, case carries with its definition the corollary that the major data items - the messages or the documents - are of rather variable structure. If the designation, Atom, is associated with the smallest unit to carry an independent security attribute, and length is

accepted as the count of atoms within the data item, then "paragraph marking" (a frequently imposed requirement involving designating security attributes for each paragraph within a document or the textual part of a message) will cause wide variability in the length of data items. This seems to imply that the structures holding these data items, or containers, are not really fixed and rigid but variable to accommodate the variety of different situations which may arise. This paper will refer, subsequently, to databases of types A and B: the rigidly, structured ones and the more fluidly structured ones respectively.

During the time which elapsed between the original studies and the Woods Hole gathering, some manufacturers (notably Intel and Honeywell) proposed architectures involving the notion of capability. Essentially, a capability is a property (or, in many cases, a possession) of a user. It enables him to access a particular datum or collection of data and may further determine what he may do with the data so accessed.

It is necessary to delineate the difference between non-discretionary and discretionary security policy. The former (sometimes called mandatory security policy) says that information may not be permitted to flow to those persons, or computer locations, not authorized to receive it. In this context, authorization usually implies the matching of security attributes (clearances for users and classifications for data) to determine that those of the recipient or destination are "higher" than those of the sender or source. "Higher" is not usually interpretable on a linear scale (e.g. Secret is higher than Confidential), but is more frequently equivalent to the notion of dominance in a partially ordered set (or lattice). In later parts of this paper we shall say that one security attribute "dominates" another rather than that one is higher than the other.

On the other hand, discretionary policy usually carries with it the idea that the contents of the item or items to which it is applied have also to be given importance in determining access rights or privileges. This follows much more closely the concept of need-to-know. Its purpose is to screen out from the class of users whose attributes dominate those of the data all but the (usually very small) subset who really do have a legitimate requirement to access the data.

Although capabilities may be used to assist in enforcing non-discretionary access (most simply by denying the capability associated with an object to those who, a priori, do not have dominating security attributes), they seem to have more noticeable utility in the discretionary arena. Setting aside the non-discretionary issue, individual users requiring different forms of access to an object may be given finely honed capabilities.

Apart from capabilities, the principle means of handling discretionary access in databases has been the access control list (ACL). This explicit enumeration of users (or roles which users can play), and the nature of permitted access associated with each, is usually tied directly to objects. Immediately implied is the granularity of access control - the grain is the data item which has its own ACL. Traditionally, these have been quite large objects.

The summer session at Woods Hole highlighted the requirement to take as the access controlled grain a data element within a relation (thus unreasonably proliferating the number of ACLs required if this technique were to be employed); it also brought into focus the fact that, very frequently, accesses may occur only subject to certain external conditions. Both reading and writing data resident in a database may be subject to this conditional access. Moreover, these conditions

are usually content sensitive (CS): they refer to the current values of the data.

Conditional writing is, perhaps, more clearly recognized. The internal consistency of the database (sometimes called database integrity) must be preserved. To achieve this consistency, and for a host of other reasons as varied as the nature of data housed in databases, the concept of side constraint has been proposed (quite independently of all security considerations). The purpose served by a side constraint is the inhibition of database modification or augmentation if such a change would violate an explicitly defined condition - the constraint.

On the other side of the problem is the requirement, being more acutely felt because of the study given it by the third group at Woods Hole, that certain data reads ought to be inhibited or controlled because they constitute violations of constraints enunciated to prevent information leakage by means of aggregation or inference. Conditional reads may be the only feasible answer to these very difficult issues.

As the result of these considerations, a new and somewhat different approach to data access within databases is being proposed. This new approach is the Rule. Borrowing from both the earlier concepts of ACL and capability, the Rule is, in essence, neither. It provides, in a fundamental way, for the conditional access of any subset of the database.

The remainder of this paper discusses the nature of a Rule, the issues raised by the idea of Rules, and, finally, some conclusions about this concept.

WHAT IS A RULE ?

The basic definition of a Rule is a 4-tuple:

R = <U,D,O,C>

where U is a list of user identifications, user roles, or couples <Id,role>
D is a subset of the database (defined context free)
O is a set of operations which may be performed on D by U
C is the set of (context sensitive) conditions which, jointly, must be satisfied before U does O to D.

The list of users (roles) is flexible as indicated. In those situations where each user is usually designated only by his identifying tag (i.e. name, machine, or number) U will contain just these tags In systems where roles are pre-eminent (for example, in tactical situations where the operator of a piece of equipment is important regardless of who is there at any moment), these alone may suffice. Newer systems, for example the Military Message System, may require the pair consisting of user identification and the user's current role for unique and unambiguous determination of access requirements. In such cases, the list may consist of pairs.

The subset of the database designated as D may be any (proper or improper) subset. Although the database is considered to be relationally organized, it is not intended that the relational operators necessarily be used to define the subset, D. Because it will usually, though certainly not necessarily, be the case that each Rule will be associated with a particular task or subtask in some on-going activity, there may be suitable pre-defined conceptual subschema (views)

associated with these tasks. Whatever method of defining the subset seems most appropriate ought to be employable, at least in the first instance - with one major caveat: the definition of D must be achieved in a totally context-free (CF) way. This implies that no reference to the continually changing data may be made at this stage; all context sensitivity is handled by the conditions, C. This is the principal reason that D (for "database subset") was used to designate the second element of a Rule instead of V (for "view", implying use of all the relational operators equally).

In practice, D is more likely to refer to fields or domains than to tuples (projection rather than selection) while C will handle the contents of the tuples. Databases of type B (those with fluid structures) will usually have wider ranging subsets defined in D; this is because the structure becomes fluid, and therefore difficult to describe in context free terms, specifically because of its contents.

The operations which might be invoked over a subset of a database are, when examined at a low level, very few in number. Complex, user-friendly commands will usually translate into a number of primitive operations. These primitives, rather than the more complex (and meaningful) aggregates of them, form the domain from which O is drawn. In an implementation of Rules, it is expected that groups of Rules will be used conjointly to permit users to perform their tasks over their subschema.

The domain of O is {read,replace,add,delete}. These activities refer to tuples. It will be argued that a suitable structure within the database makes this domain, slightly augmented, sufficiently powerful to permit all the usual activities to be controlled in a satisfactory manner. This is true for even the more application specific situations.

It is in the area of conditions that major difficulties arise. Any software written to use Rules to mediate access to a database must have, as a significant part of it, a condition interpreter. By their very nature, conditions are context sensitive. They are predicates which are either true or false, based upon, in most cases, the present state of the data in the base. Not infrequently other entities besides those found in the database must also be referenced; these may include the system clock or other elements of the operating system domain, or data items intended for introduction to the DB. For the required access to occur, the conditions must be checked at the time the operation is requested.

If, indeed, interpretation of the condition is to take place *in tempo*, the question arises as to how it is to be expressed to expedite not only its interpretation but the execution required to determine its truth value. This question, to some degree, harkens back to the previously raised concern about the definition of the subset, D; it is precisely here that the context sensitive relational operators do, indeed, come into play. The fact that entities not in the DB must also be referenced indicates, however, that these operators are not sufficient.

Another obvious difficulty is the possibility that conditions might refer to parts of the database to which the user is denied access for reasons arising from non-discretionary policy. The potential for information leakage here seems very great. However, in the following discussion of issues it is argued that this channel may be blocked by using suitable techniques. Recognition of this problem is essential as a first step in controlling it.

DISCUSSION OF ISSUES

In order to test the feasibility of Rules as the basis of access control in DBMS, the Military Message System Security Model was chosen as the example system. This system addresses the notion of multi-level entities; that is, atoms and containers interact to form a hierarchy. As well, the second class of database, the textual stream, at Woods Hole took the same model as its starting point and produced a tentative paper design for a secure document system.

Applicability Issues

The present section will explore points that are applicable to type B databases, which are essentially textual, and consider the degree of applicability each may possess to those databases which are more rigidly structured and essentially numeric.

The container/atom model developed in the paper [Bonyun 83] indicated that the Rule concept is sufficiently powerful to handle the type B databases, which are more fluidly structured than type A. Implied are two related beliefs: (1) that type B is more general and more difficult, and (2) that if our concept of Rules can be used here there is every reason to believe that it can be employed to advantage in simpler, more conventional cases.

Five generic relational forms were assumed to form the basis of the database system. These are Atom, Link, Container, User, and Rule. While there is some question about the direct applicability of these ideas to type A databases, it ought to be remembered that besides message systems, document handling systems and word processing applications fall into type B; consequently, they are subject to the same kind of considerations. We take it as axiomatic that all type B databases, besides being largely textual, have the multilevel requirement realizable by the container/atom model.

Three points were derived from the example:

1. Distinction between reading and using an existing data item

The use of a data item for arithmetic purposes is nearly the same thing as the use of an existing sentence in building a new paragraph. One can imagine instances of data being available for output purposes alone and being forbidden as direct computational elements. Because of this, the list of operations, O, in Rules ought probably to be permanently augmented by use.

2. Actions within Conditions

A small percentage of Rules seem to require some modification to existing data, as well as the determination of the truth of the predicate, in order that the operation on the data be permitted to proceed. This action constitutes a side effect. Perhaps consideration ought to be given, likewise, to the idea that the basic definition of Rule be augmented to include a fifth (optional) part, Actions.

3. Universal and Specific Rules

Two kinds of Rules emerged - the universal and the specific - those which handle the overall system and its individual elements, respectively. The use of universal Rules was shown to be useful as a means of enforcing the policy, or assertions given by the security model. These assertions constitute the heart of the model; the universal Rules, if enforced, assure their validity. The recursive nature of Rules (Rules which talk about Rules) may also be a useful notion when universal Rules are being developed for security models.

Specific Rules are the chief means of enforcing, and even stating, discretionary access control; the Rule is, primarily, a technique with the ability to manage discretionary access limitations to data. In the area of very finely granulated discretionary access control, specific Rules are the appropriate vehicle.

Theoretical Issues

Two problem areas are related respectively to the D and C parts of the Rule. Questions arise about the various methods which might be employed to define the subset of the database, and about the nature and required power of the condition interpreter.

Database Subset Definition

The design of a logical substructure and the consequent thorough knowledge of it will materially assisted the writing of Rules.

If access control is to be at the data element level, then not only must the non-discretionary attributes be attached to each data element, but discretionary access must also operate at this level. As the Rule is intended to be the primary vehicle for this control, it must have the capacity to designate any collection of data elements for this purpose. It is for this reason that we believe the usual relational operations are not sufficiently powerful. The result is the contention that the definition of D ought not to have to rely solely on relational operations.

When collections of primitive operations are required to perform a single logical task, the definition of D is again important. Many Rules are likely to be involved (and sequentially ordered), and two external conditions must be satisfied:

1. the extension of the complex operation over the database must be shown to lie totally within the D of the Rules; and

2. all the operations in the sequence must be treated as a unit for purposes of timing: other uses of the subset D must be shut out for the duration of the activity.

It is desirable to have, as part of the complex operation definition, any global internal consistency check which may seem required as well as a way of backing out while still in the critical zone, if this becomes necessary because the check fails. The use of well structured type managers to define complex

operations would appear to be the best way of accomplishing all this.

The Condition Interpreter

The condition interpreter (CI) must have sufficient power to be able to ascertain the truth or falsity of the predicate parts of the conditions and also to effect the required actions. The nature of this power is the understanding of the variable names used in the statements of the conditions and the capacity to do the requisite computations involving them.

The following overall conclusions have been drawn about the nature of the CI:

1. It must be efficient, and this efficiency will likely be achievable only if a suitable formalism can be found; while nothing has been said about candidate formalisms, the author's prior experience with APL makes this seem to be a very suitable potential medium for the expression of conditions, particularly as APL has a number of very efficient interpreters already written for a variety of hardware including microprocessors; and

2. It must be able to access not only every part of the database, by direct address and by content, but also a wide variety of other data items outside the database; a full enumeration of these outside items is desirable in any particular implementation.

Security Issues

Prior to any attempt to deal with security matters, it is possible to hypothesize that every part of the database ought to be accessible to the CI at all times. We must now explore the consequences of the fact that every data item within the database, and the outside data elements also known to and accessible by the CI, all have their own individual security attributes.

The real issue is the fear that the CI, in order to check a condition predicate, will access subsets of the database which are outside the normal access rights of the user requiring the checking. A conflict is imminent between the requirements of database integrity and non-discretionary access.

Rather than have to choose between the two, we are suggesting that a solution exists which simultaneously satisfies both requirements. The solution proposed permits the unrestricted use of Rules and their conditions without fear that a leakage path is thereby being forged. We claim that there does not exist any channel back to the user about data in the environment to which he does not have legitimate access.

The proposed solution is to hide the conditions (in fact, the complete Rule) from the user. When presumed to be a part of the database, a rule has its own security attributes. These may be forced to be the least upper bound of the attributes of any data mentioned by them. In conjunction with the absence of any specific Rule permitting anyone to read or use Rules, this assignment of attributes is almost enough. Missing is the capacity, assumed to be present in the general case, of users making Rules to embody the discretionary aspects of security.

The puzzle becomes solved if one realizes that the problem of conditions referring to unauthorized data elements arises mainly in the area of database integrity. This is a system problem which almost always requires universal Rules. Discretionary access, on the other hand, the main use of Rules which are presented and installed by users, ought not to have such far-reaching conditions.

The price to be paid, therefore, seems to be to permit the creation of Rules involving data items of a higher classification only to those possessing sufficiently high clearances. In the case of integrity Rules, this probably means either the SSO or the database manager, both of whom are expected to have clearances which dominate every data element.

The only remaining issue involves what the user will be told when a condition fails. It would seem that the best course of action is to tell the user as little as possible: nothing more than that the requested activity cannot be completed. If there are a number of adjacent primitive accesses contingent on this success, then this simple failure condition must be recognizable by the type manager or other subprogram defining the complex operation; any required backing out should then be begun as a consequence.

Without any knowledge about the various elements in the conditional part of the applicable Rule(s), or even which Rule may have failed, the user cannot possibly make any inferences. All he/she will know is that the proposed activity cannot proceed. Multiple attempts may be tried; but even if a later one is successfully completed, there is no inference possible because the reasons for earlier failure are not known.

Implementation Issues

The location and availability of Rules and certain other design decisions will need to be considered in order to make the invocation of the Rules as smooth and rapid as possible. It seems to be essential, if Rules are to be handled expeditiously and cheaply, to have handy an enumeration of those Rules which will have to be invoked so that excess or repeated "seeking" within the database is not required.

It would be a timesaver if the Rules themselves, and not just their identification, were readily available without a retrieval from the database being necessary; but, as a general case, this will probably be impractical and each database access will require one or more Rules to be fetched and sent to be interpreted. If there is some form of faster peripheral memory available, it seems clear that the Rules ought to reside therein.

As has been found in other areas of computer security, it is apparent that a system to employ Rules must be designed, from its beginnings, with this in mind. Retrofitting Rules to existing systems seems to be impractical and undesirable.

Issues in Distributed Systems

When data (and/or processing) is physically remote, there are time delays. If the notion of criticality is extended to systems which are distributed, then the delays which one must expect become crippling. Locking out other users while a collection of primitive operations takes place is conceivable only if the period

of time of lockout is small. When all required data elements are close by, this is probably reasonable to expect. When the elements are scattered over a distributed system, no such reasonable expectation can be made.

Clearly, this problem is not unique to Rule based systems. What makes the matter worse for such systems is the very general nature of the conditional part of Rules. The CI must be able to access a great many data elements in order to be effective. When these elements are widely distributed, the picture becomes intolerable.

A number of implementation strategies may be suggested to help alleviate the problem; the design of distributed systems in general has enjoyed a great deal of study [Lampson 81]. Yet, until some of the basic problems associated with distributed systems are completely under control, the extension of the notion of Rules to such systems would seem to be best deferred.

CONCLUSIONS

The conclusions which may be drawn from the discussion above are largely the result of personal and subjective analysis of the many problems and issues raised (and of some not made explicit). The following short list is a synopsis of the main ideas held by the author: readers are encouraged to differ in their conclusions and to communicate their differences to the author.

1. A Rule seems a suitable generalized way to manage both discretionary access and the controlling aspects of specialized systems such as the MMS

2. The security issues seem to be under control if the Rules are, in general, hidden from the user and if only minimal information is given back to the user in the event that a Rule has caused an access to fail.

3. The primitive operations and the observation that user activities will usually employ several of these primitive operations collected into clusters (handled by a type manager) raised the idea of critical regions of code so that database integrity be maintained. The extension of this concept over distributed systems seems to be a particularly difficult task requiring more time and results from research into distributed systems.

All in all, we believe that Rules are powerful tools which have a legitimate place in the future design of secure database systems.

REFERENCES

[Bell-LaPadula 75] Bell, D.E., and LaPadula, L.J., "Secure Computer System: Unified Exposition and Multics Interpretation," M74-244, MITRE Corp., Bedford, Massachussets, July 1975.

[Bonyun 83] Bonyun, D.A., "Rules as the Basis of Access Control in Database Management Systems," TR-83-5060-01A, I.P. Sharp Associates Limited, Ottawa, Canada, June 1984.

[Denning 76] Denning,D.E., "A Lattice Model of Secure Information Flow," Commun. ACM19(5), 236-243 (May 1976).

[Grohn 76] Grohn, Michael J., "A Model of a Protected Data Management System," ESD-TR-76-289, I.P. Sharp Associates Limited, Ottawa, Canada, June 1976.

[Hinke 75] Hinke, T.H., Schaefer, M., "Secure Data Management System," RADC-TR-266, Rome Air Development Centre, AFSC, Griffiss AFB, N.Y., Nov. 1975, (NTIS AD A019201).

[Landwehr 82] Landwehr, C.E., Heitmeyer, C.L., "Military Message Systems: Requirements and Security Model," NRL Memorandum Report 4925, Washington D.C., 1982.

[Lampson 81] Lampson, Paul, M., Siegert, H.J., (Editors), Distributed Systems - Architecture and Implementation; Advanced Courses," Springer (Berlin-Heidleberg) Verlag, N.Y., 1981.

Structure of a Rapid Prototype Secure Military Message System

Mark R. Cornwell
Robert J. K. Jacob

Computer Science & Systems Branch
Naval Research Laboratory
Washington, D.C. 20375

ABSTRACT

Past attempts at building multilevel-secure systems have resulted in human-interfaces that were difficult to understand and use. We posit that part of this difficulty results from a poor fit between conventional security models and the intuitive notion of security users apply to their application. The Secure Military Message Systems project attacks these problems by defining a security model intuitively closer to the application and testing this model by constructing rapid prototype systems and trying them out. Techniques used to construct these rapid prototypes include the definition of abstract data types, an intermediate command language, and an executable formal specification of the human-interface. Features of the MMS security model are presented using examples from a rapid prototype system.

1. The Problem

The Secure Military Message Systems project is building rapid prototypes in order to learn about techniques for building secure computer systems. In the past, secure systems have been built from general-purpose security models. While this yields an internal model that is elegant and easy to understand, the user interface of the resulting systems is often confusing, because it enforces security restrictions that appear counter-intuitive from the perspective of a user. Our solution to this problem is to define a security model that attempts to capture the user's intuitive notion of security in a military message system [1]. Then, we examine the effects of that model on the system behavior visible at the user interface by building and studying a series of rapid prototype systems that implement the model. A sample session with one such rapid prototype is given in the appendix. We have designed message systems offering a representative range of functions for composing, reading, distributing, and processing military messages. While the present security model was motivated by message systems, it has been found to be adaptable to other similar types of document systems.

Our definition of security is embodied in a security model for military message systems. This model is described in detail by Landwehr [1]. The MMS security model consists of a set of definitions, assumptions and assertions. It differs from some conventional models of security such as that of Bell & Lapadula [2] in that it directly models multi-level objects, such as a SECRET message containing CONFIDENTIAL paragraphs; and recognizes message system operations such as RELEASE, rather than the generic READ, WRITE, EXECUTE.

2. Design Goals of the Rapid Prototype Systems

The rapid prototype systems are intended to exhibit the user-visible behavior of secure message systems conforming to the MMS model. To make our prototyping effort feasible we have chosen to concentrate on the security model, functional requirements, and user interface. We have tried to implement these aspects of the design faithfully, while other concerns that could be important for a production system are not addressed. For example, time and space efficiency are not a concern as long as they are acceptable for demonstration purposes. The rapid prototypes support only a few users and a low volume of message traffic. Genuine security was not addressed, but the normal behavior of the rapid prototype is just like that of the corresponding secure system. Some other concerns are important for the rapid prototypes but not necessary for a production system. For example, the rapid prototypes themselves should be designed and built quickly. They should be easy to modify to reflect changes both in functionality and in the underlying security model. At this stage, obtaining a genuinely secure implementation is not as important as obtaining an apparently secure one quickly.

The software decomposition of the M2 rapid prototype (earlier prototypes were called M0 and M1 [3]) does address problems of building a system that satisfies the MMS security model. The internal design has incorporated lessons learned over several generations of rapid prototype systems. In M0 and M1, security checks were widely scattered throughout the code. The decomposition we present here identifies and localizes many of the mechanisms that enforce the MMS security model. This locality increases the promise of a verifiable implementation of the design.

3. Structure of the Rapid Prototypes

The M2 system is partitioned into two components, a user interface component and a semantic action component as shown in figure 1. The user interface handles the details of transforming sequences of keystrokes, mouse clicks, and other user input into requests in an Intermediate Command Language (ICL) and presenting the results of such requests as output to the user. The ICL requests themselves are processed by the semantic action component.

The user interface component is specified as a set of state diagrams after the manner described by Jacob [4]. In the state diagram model, an automaton reads from a stream of tokens and makes a transition to another state based on the token read. Actions may be associated with state transitions; whenever such a transition is made, its associated actions are performed. In the present system, the actions consist of ICL commands, which are transmitted to and executed by the semantic action component. The user interface component is implemented by an interpreter that executes the state diagram specifications [5]. It traverses the diagrams and performs the actions associated with each transition.

This division permits new systems with different user interfaces to be constructed from the existing system conveniently. If the design of the user interface is changed, only the user interface component must be modified so that it will translate from the new command language into the same ICL commands; the semantic component of the system need not be changed. The division also provides a useful decomposition of programming tasks. Given a stable description of the ICL, the user interface and semantic components of the system can be developed in isolation from one another, since the only communication between them is via the defined ICL commands. In fact, the M2 prototype was coded in just this fashion, by the authors working in parallel. Our experience showed

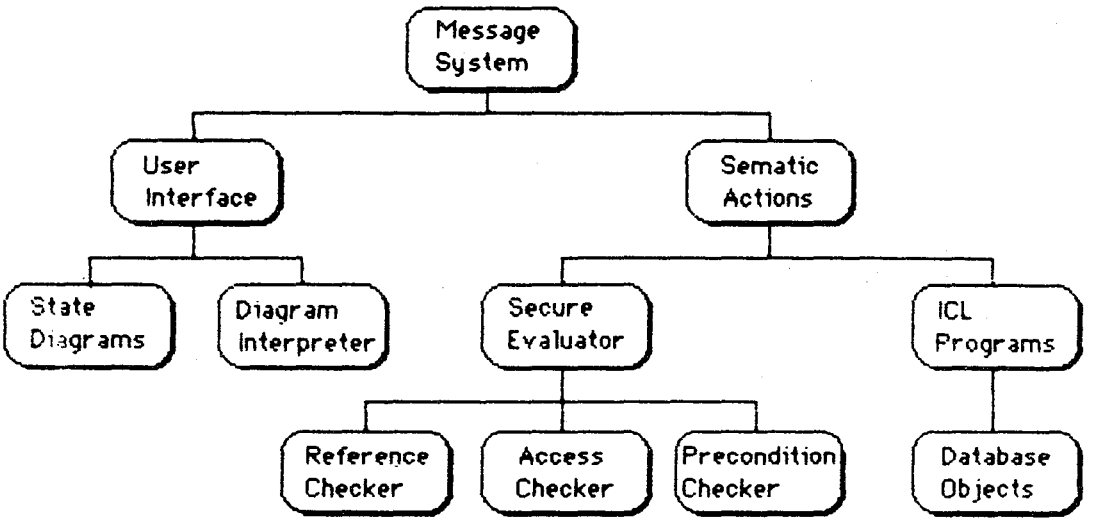


fig. 1. System Decomposition

that, with this decomposition, it was possible to make major changes to either component of the system without affecting the other component or even telling the person writing it.

The user interface itself is designed to be easy for naive users to learn (in contrast to being easy for experienced users to operate). A command typically consists of a verb (such as Display), an object (such as "the file named inbox"), and, possibly, some extra parameters (in this example, a display filter). The verb is selected from a menu; objects and other parameters are selected from menus where possible or else typed in a window. For each item, a default value is available with a single keystroke. After the user interface component has accepted an entire ICL command with all its arguments and found it to be syntactically correct, it issues the complete command to the semantic component. Finally, there are some user-level operations that do not correspond to ICL commands, such as the commands to abort a command, scroll the windows, select special-purpose menus, and exit. These are placed on function keys and may be entered by the user at any point in the dialogue.

The semantic action component of the system is itself partitioned into a secure evaluator and a set of ICL programs. The ICL programs perform the actions requested by the user. The secure evaluator ensures that ICL programs never get a chance to perform actions that would violate security. It performs all the security checks necessary before an ICL program is invoked.

The actions taken by ICL commands are the user's means for inspecting and manipulating the sensitive information in the system. Each command available to the user corresponds to one of these ICL programs. The ICL programs manipulate objects in the message system database, such as directories, message files, citations, and messages, which embody the sensitive information in the system. ICL programs are written in terms of operations on these lower level objects. Unlike the ICL commands, the user does not have access to these lower level operations directly. Users can only invoke them indirectly by invoking ICL commands.

The secure evaluator is made up of programs to perform three kinds of security checks. The reference checker enforces security constraints while determining what objects in the system a requested ICL operation would act upon. The access checker compares the access permissions of these objects with the privileges of the user supplying the request. A precondition checker uses its knowledge of the semantics of the ICL programs to determine if a request should be denied or allowed to proceed.

In the interest of building the system rapidly, we decided to use an existing text editor, Emacs [6], for composing and editing messages. Emacs is an extensible text editor with its own language for defining new commands and modifying its user interface. We took advantage of this extensibility to tailor it to approximate a secure editor for messages and to integrate it into our rapid prototype. It protects some fields of a message from modification and it provides some prompts and syntax checking on messages.

4. Abstract Types, Inheritance and Locality

Most of the implementation is oriented around abstract data types. Our notion of abstract data types is fairly conventional but includes operator inheritance and overloading concepts similar to those of Smalltalk [7], and Flavors [8]. An abstract data type characterizes a class of data by associating a type name with a set of values and a set of named access operators. The access operators may alter values, return information about values, or both. A type may be a subtype of another type. If *B* is a subtype of *A*, then by default *B* inherits the access operators of *A*. *B*'s definition can then add new access operators not associated with *A*. By overloading operator names (defining new operators with the same names as inherited operators) *B* can hide inherited operators.

This notion of types, and operator inheritance in particular, helped in implementing security. A type ENTITY was associated with the security specific information (e.g. classification, access set, CCR mark). Other types such as MESSAGE were defined as subtypes of type ENTITY. The security checking programs were written for ENTITY, and thus didn't depend on specifics of type MESSAGE, only type ENTITY. We found it possible to add new secure types to the system by defining new subtypes of ENTITY without adding new security checking programs.

Our design uses abstract data types to define a wide variety of data classes. Some are visible to users: ENTITY, ACCESS_SET, MESSAGE, MESSAGE_FILE. Others are used internally: STATE_DIAGRAM, TRANSITION, REQUEST.

5. Security Model and Mechanisms for Security Checking

This section describes some specific techniques used to implement the MMS security model in the semantic component of the M? prototype. Our presentation of the model will be in terms of the techniques we have chosen to implement it, though other implementations could satisfy the model without using these particular techniques. Concepts from the model will be briefly described as necessary. A complete definition and explanation of the model is given by Landwehr [1].

The MMS security model provides a set of definitions and assertions characterizing a secure system. *Entities* are units of information in the system that are associated with protection information. Every entity has a *classification*, an *access set*, a *type* and a *value*. *User ID's* represent the human users of the system. Every user ID has an associated *clearance* and a set of *roles*. Users indicate the entities they access by providing *references* to them. A *direct reference* (e.g., MSG1190) is an atomic identifier that denotes exactly one entity

independent of the values of any other entities. Entities may *contain* other entities. An *indirect reference* indicates an entity by referring to an entity that contains it. (e.g., "the fifth message in Cornwell's inbox file"). Say, an entity *e1* contains an entity *e2*. If *e1* is marked *container clearance required (CCR)*, then a user can't use *e1* in an indirect reference to *e2* unless the user is cleared for *e1*.

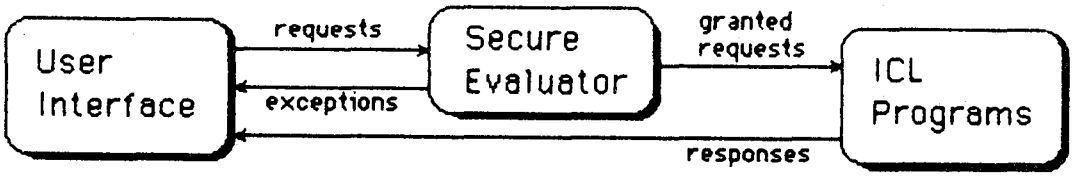


fig 2. Security Mediation

The division of the semantic action component into a secure evaluator and ICL programs separates security checking from the normal case semantics of user commands. The secure evaluator mediates the access of the user interface to the ICL programs as shown in figure 2. The user interface constructs ICL requests and sends them to the secure evaluator, which either grants the request, by invoking an ICL program, or sends an exception back to the user interface. After a request is granted, the requested ICL program can run to completion without the need to perform any other run-time security checks. This obviates the need to consider problems that occur when a operation encounters a security exception after it has begun its execution, such as restoring state or translating low level security exceptions into meaningful responses to the user.

A request that enters the secure evaluator must pass three kinds of security checks before it is granted. Each kind is handled by a different component of the secure evaluator, these components being a reference checker, an access checker, and a precondition checker. A request *<op r1 ... rN>* consists of an operator followed by a sequence of references and possibly other parameters. The requests goes through the following checks:

1) *Reference Checker*. Any violation of constraints imposed by CCR marks is detected at this step. Each of the indirect references is dereferenced. This yields a structure of the form *<op e1 ... eN>* where the *ei* is the the entity denoted by *ri*.
2) *Access Checker*. For each entity *ei* in *e1...eN* we check to see that the access set of the entity permits the user to apply the operator with that entity as its *i*th parameter.
3) *Precondition Checker*. Finally, we perform a check to determine whether performing the requested actions the current state will maintain the security assertions. If so, we apply *op* to *e1...eN*.

Failure to pass any one of the above checks will cause the request to be denied and a security exception to be generated. A brief explanation of why the request was denied is passed back to the user interface which conveys it to the user.

With this outline of run-time security checking in place, we examine each

component of the secure evaluator elaborating on the checking each performs.

The reference checker evaluates the references in an ICL request based on the classifications, values, and types of the entities appearing in each indirect reference. Every type of entity that can contain other entities is associated with a selector function S that given an entity and an index returns the entity contained by the given entity at that index. For example, if *e* denotes a directory of message files and *i1* is the name of a file in that directory, then *S(e,i1)* denotes that file. Each indirect reference *<e,i1,...,ik>* is a sequence whose first element is an atomic identifier for an entity and whose remaining elements are indices. There is a procedure for evaluating a reference to determine the entity it denotes. This dereferencing procedure maps references to direct references. It acts as an identity on direct references. For indirect references it replaces the first two elements of its argument with the entity denoted by applying a selector function to e and i1, and applies itself recursively to the result.

The dereferencing procedure also performs some security checks and will generate an exception if attempted violations occur. An exception will occur if the reference is indirect, the entity heading that reference is marked CCR and the user's clearance does not dominate the classification of that entity. Notice that it is possible to dereference an indirect reference that depends on entities for which the user is not cleared without generating an exception.

The access checker compares the access sets of the requested entities with the operator name and the privileges of the user making the request. *Operators* are the commands users may invoke (directly) to change or inspect the entities in the database. In the M2 rapid prototype, these operators are defined to be the ICL programs. The access set of an entity determines what operators may be applied to it and by what users. An access set for a given entity *e* is a set of triples of the form (userID or role, operator, k). A user can use entity *e* as the *k*th parameter of an operator if a triple with the user's userID (or one of the user's roles), that operator and *k* is in the access set of *e*.

The precondition checker compares the state of the system with the requested action to determine whether the action can take place without violating any security assertions not already addressed by the reference checker and access checker. One such assertion is the hierarchy assertion, which states that the classification of every entity must dominate the classification of every entity it contains. For example, a message file contains a set of citations and each citation contains a message. The message file must be classified higher than any of its citations and each citation in turn must be classified higher than its message. Another assertion states that only a user acting in the role of system security officer can change the clearance associated with a user ID.

In order to guarantee that such assertions remain invariant, every operator *op* is associated with a precondition *pre(op)* characterizing the conditions under which applying *op* will leave the system in a state satisfying the security assertions. Before applying any operation, this precondition is checked and an exception is generated if the precondition does not hold. Assuming the preconditions are correct, an operation never causes an action that will invalidate the security assertions.

Preconditions are attractive because mathematical techniques (weakest preconditions, predicate transformers [9] ,Hoare logic [10]) exist to derive them from a specification of the semantics of the operator and a specification of the security invariants. Dijkstra [9], Gries [11], and others have argued convincingly that deriving programs from specifications yields benefits that verifying programs after they are written does not. In deriving precondition checks for our rapid prototype, we applied the concepts of invariant assertions and weakest

preconditions informally to specify and program the system. Using this approach, we could then attempt to provide formal proofs that preconditions are sufficient to insure that the operations maintain the security invariants.

The use of an external text editor poses some problems to our security design. Within the message system, messages are represented in linked structures laden with security information. A message sent to the editor is translated into a text form visually familiar to users but with much of the security information (e.g. access sets) stripped off. The system should prevent users from editing certain fields, corrupting the security labels on entities, or entering ill-formed messages into the system. To this end, we partition the message being edited into an editable and noneditable part, displaying the latter in a window where the user cannot modify it. In Emacs, the user edits a textual representation of a message in the absence of any security checking. When Emacs exits, the message system parses the message, checking it for well-formedness and conformity with the security constraints before we allowed it to be stored in the message system data types. For example, a message might fail the well-formedness check if a user mistypes a field name or leaves off a security label on a paragraph. The message would not conform to security constraints if, say, a CONFIDENTIAL text field held a SECRET paragraph.

6. Future Directions

The rapid prototype described here is one of a series of systems being built to investigate secure message systems. The next steps in this work include:

a) Obtain feedback from real message system users using the M2 prototype.

b) Develop a user interface incorporating a bitmapped display, graphics and mouse. This will be done by modifying the user interface component (and moving it to a different host), while leaving the semantic component unchanged.

c) Investigate formal techniques for deriving precondition checks from the specifications.

d) Build a genuinely secure full-scale prototype based on the current rapid prototype.

7. Summary

We have observed that conventional security models, while intuitively appealing to designers, can appear confusing and inappropriate when viewed through the user interface of a finished system. To overcome these problems, a specific security model has been defined to conform to an intuitive notion of how the user interface to a secure message system should behave. A series of rapid prototype systems has been built to allow us to observe directly the interactions between this security model and a user interface. The M2 rapid prototype demonstrates a particular approach to the implementation of a system that incorporates this security model. The techniques used in building it are applicable to other application based security models.

8. Acknowledgments

Carl Landwehr, Connie Heitmeyer, and John McLean developed and formalized the MMS security model. Jean Tschohl, Ken Pon, and Brian Tretick contributed to building and testing the rapid prototypes. This work was supported by the Naval Electronic Systems Command under the direction of H.O. Lubbes.

The following scenario, based on the M2 prototype, illustrates some fundamental ideas of the Secure Military Message System Design. In this example a user, Jones, logs onto the message system and reads some incoming mail. The session illustrates some of the data objects the users manipulate and how message processing is integrated with the security policy.

To gain access to the system, Jones must first log in. Jones does this by providing a userID, the classification that the screen is to assume, in this case (T cnwdi nato crypto), and and a password. A menu appears on the screen from which Jones selects active roles for the session. The system checks to see that a user with userID Jones and the given password is authorized to use the system, and that Jones is authorized for each of the roles selected. With this precondition satisfied, the login operation proceeds.

The screen has a current classification at the level specified at login. Citations from Jones's message file "inbox" are displayed on the screen (*fig. 1a.*)

```
Display Message File inbox
-------------------------------------------------------------------------------
 DISPLAY | CREATE   | DELETE   | UNDELETE| COPY  | MOVE  | EXPUNGE | EDIT
Msg/File/| Msg/File/| Msg/File/|   Msg   | Msg   | Msg   |  File   | Msg/Text
Text/Dir | Text     | Text     |         |       |       |         |
-------------------------------------------------------------------------------

1 N  (U)
     From: (U) Dwork        Subj: (U) Ada Conference

2 N  (SECRET cnwdi crypto)
     From: (U) Adams        Subj: (S) Beethoven Combiner

3 N  (CONFIDENTIAL cnwdi nuclear)
     From: (U) JPL          Subj: (C) Dense Pack Simulator

4 N  (UNCLASSIFIED)
     From: (U) NSA          Subj: (U) Security Evaluation Standards
```

fig. a1.

At this point the screen (an entity) contains a sequence of citations (also entities). Each citation contains the From, Subject, and Security fields of the message to which it refers. Only citations below the the classification of the screen are displayed.

Displaying the second message, Jones can see all of its fields (*fig. a2.*) Notice that the message classification dominates the classification of each of its fields. Similarly, the text field classification dominates the classifications of each of its paragraphs.

```
Display Message inbox 2 all
-----------------------------------------------------------------------------
 DISPLAY  | CREATE   | DELETE   | UNDELETE | COPY  | MOVE  | EXPUNGE | EDIT
Msg/File/ | Msg/File/| Msg/File/|   Msg    |  Msg  |  Msg  |  File   | Msg/Text
Text/Dir  |  Text    |  Text    |          |       |       |         |
-----------------------------------------------------------------------------

Security: (S cnwdi crypto)

From:     (U) Adams
To:       (U) Jones
Subj:     (S) Beethoven Combiner

Text: (S cnwdi crypto)

(U) first paragraph

(S) second paragraph

(S cnwdi crypto) last paragraph
```

fig. a2.

Jones's directory contains all of the message files belonging to Jones. The message file names are displayed along with the file classifications (*fig. a3.*)

```
Display Directory Jones
-----------------------------------------------------------------------------
 DISPLAY  | CREATE   | DELETE   | UNDELETE | COPY  | MOVE  | EXPUNGE | EDIT
Msg/File/ | Msg/File/| Msg/File/|   Msg    |  Msg  |  Msg  |  File   | Msg/Text
Text/Dir  |  Text    |  Text    |          |       |       |         |
-----------------------------------------------------------------------------

  Cryptography        (T cnwdi crypto)
  Dense Pack          (T cnwdi nuclear)
  Misc                (U)
  Nato MRM            (C nato)
  Sensor Project      (U)
  Specifications      (U)
  Submarines          (C cnwdi nuclear)
  inbox               (T nato crypto cnwdi nuclear)
```

fig. a3.

Accidental violations of the security model are prevented. Jones is informed of denied requests in terms of familiar message system concepts. For example, if Jones attempted to save the second inbox entry (displayed earlier) into the file "Nato MRM", the system would deny the request. A brief explanation would appear in the error window (just below the menu) saying that the message was classified to high to be inserted in the given message file.

Jones may move the message in the file Cryptography, since that file's classification dominates that of the message. Jones does so and logs out leaving the terminal ready for another login.

References

1. C.E. Landwehr, C.L. Heitmeyer, and J. McLean, *A Security Model for Military Message Systems*, Naval Research Laboratory, Washington, D.C. (May 1984).
2. D.E. Bell and L.J. Lapadula, "Secure Computer Systems: Mathematical Foundations and Model," MTR-2779, Mitre Corp., Bedford, Mass. (July 1975).
3. C. Heitmeyer, C. Landwehr, and M. Cornwell, "The Use of Quick Prototypes in the Secure Military Message Systems Project," *Software Engineering Notes* **7**(5) pp. 85-87 (December 1982).
4. R.J.K. Jacob, "Using Formal Specifications in the Design of a Human-Computer Interface," *Comm. ACM* **26** pp. 259-264 (1983).
5. R.J.K. Jacob, "An Executable Specification Technique for Describing Human-Computer Interaction," in *Advances in Human-Computer Interaction*, ed. H.R. Hartson, Ablex Publishing Co., Norwood, N.J. (1984). in press.
6. J. Gosling, *Unix Emacs*, Unipress Software, Inc., Highland Park, N.J. (January 1983).
7. A. Goldberg and D. Robson, *Smalltalk-80 The Language and Its Implementation*, Addeson Wesley (1983).
8. D. Weinreb and D. Moon, *LISP Machine Manual*, Massachusetts Institute of Technology, Cambridge, Mass. (March 1981).
9. E.W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, Edgewood Cliffs, N.J. (1976).
10. C.A.R. Hoare, "An Axiomatic Basis for Computer Programming," *Comm. ACM* **12** pp. 576-580 (1969).
11. D. Gries, *The Science of Programming*, Springer-Verlag, New York (1981).

Communications System Security Evaluation Criteria

Peter C. Baker

Ford Aerospace & Communications Corporation

ABSTRACT

This paper presents justification that there are sufficient differences between operating systems and communications systems to warrant separate criteria for evaluating the security properties of communications systems. The criteria address those types of communications systems that handle sensitive information and therefore must comply with DoD security policy. The criteria require the DoD security policy be stated in terms of acceptance/delivery criteria and an internal control criteria. In addition, the proposed criteria requires, for higher assurance classes, that the design and implementation of the communications system be evaluated in terms of a systematic decomposition strategy in order to provide insight into the internal security properties of the system. A summary of the proposed evaluation criteria for communications systems is presented.

1. Introduction

The benefits of remote resource sharing between operating systems has promoted growth in the development of supporting communications systems. In cases where these operating systems must process and exchange sensitive information, additional requirements for protecting this information is placed on the communications system. Security criteria have been developed and are being applied in the evaluation of the individual operating systems and their supporting hardware bases. Because of some significant differences between operating systems and their supporting communications systems, it is not appropriate to attempt to apply operating system security evaluation criteria to these communications systems. Therefore, to insure that the total information system, composed of both operating systems and communications systems, is protected from disclosing sensitive information, appropriate security evaluation criteria for the communications systems must be developed.

Section 2 describes the bounds of the communications environment for which the criteria was developed. As motivation for the development of communications system evaluation criteria, Section 3 describes important differences between operating systems and communications systems, and provides criteria objectives for such Communications Systems. Section 4 describes the requirements for a Communications System Evaluation Criteria. Section 5 describes the proposed criteria and provides a

---

- This effort was partially funded by RADC Contract F30602-81-C-0233.

description of the assurance classes. Finally, section 6 raises some issues that result from this effort.

2. Scope

The range of communications systems can be very broad; this section establishes the scope of communications systems that are candidates for evaluation.

2.1 Layered Protocols

For the purposes of this proposed evaluation criteria, communications systems refer to message, packet and circuit switching systems that are structured according to the concepts of either the ISO or DoD Reference Model [1,2] for layered protocols. Also, the criteria is aimed primarily at communications systems that implement the lower protocol layers of these models.

2.2 Protocol Verification

The proposed criteria, at the higher assurance classes, require the formalization of certain security functions. It is expected that these functions will be implemented by various processes throughout the system. The extent to which a given protocol process implements these functions will determine the amount of security verification commensurate with the assurance class.

2.3 Security Functions

The criteria address data compromise, data integrity, denial of service, audit, accountability and authentication as security functions. Of these functions, the protection of sensitive information from compromise must be modeled by formal methods, at the highest defined assurance class.

2.4 Operating System

By the term operating system is meant a multi-user, time-sharing operating system in a single-host environment.

3. System Differences

The differences between operating systems and communications systems are sufficient to motivate the effort to develop a communications system evaluation criteria. These differences are discussed below.

3.1 Distributed Structure

Operating systems are not necessarily distributed. Communications Systems are, of necessity, distributed systems. Communications system components can be communications systems in their own right, and each level can exhibit a broad range of functional, performance and physical characteristics. These characteristics are in contrast with operating systems, which are typically implemented in single-host environments. Most single-host systems have been developed with a single abstract view. This single abstract view is not appropriate in communications systems when attempting to understand and model their security properties. Multiple layers of abstraction are essential to understand the security properties and security implications of such systems and their myriad components.

CRITERIA OBJECTIVE: The capability to evaluate the security properties of designs and implementations that represent distributed systems displaying a range of functional, performance and physical characteristics.

3.2 Protocol Structure

There are significant protocol structure differences between operating systems and communications systems. Operating systems implement more of the upper protocol layers in terms of the DoD or ISO Reference Model, and they are all end-protocol functions. Communications systems implement generally only the lower layer protocols, and these tend to be both link and switching protocol functions. Operating systems also tend to treat these protocol functions peripherally and consider them as one entity. Protocol functions are central to a communications system, and are a major factor in a communications system architecture. Although fewer layers are supported, more functions within a given layer are present, due to the possibility of different protocol sets implemented by different hosts. Functionality, not present in hosts, includes protocol switching and routing as well as various conversion functions to allow host interoperability.

As in the argument for distributed systems, single abstract views of protocol structures and hierarchies are no longer appropriate. Characteristics of these protocol structures include the concepts of protocol layer hierarchy, protocol function independence and protocol layer hiding. Components of a communications system may implement different protocol suites, depending upon the host operating system they support. Due to the hierarchical nature of the protocol architecture, some nodes may be hidden from other nodes at a given protocol layer. Again, multiple layers of abstraction are essential in attempting to understand the security properties of existing and anticipated protocol architectures.

CRITERIA OBJECTIVE: The capability to evaluate the security properties of designs and implementations that represent hierarchical protocol structures.

3.3 Internal Security Policy

In typical operating systems the security policy seen by the users is the same as the internal security policy. Communications systems may be required to implement different internal security policies. For example, certain protocols do not support sensitivity labels in their headers, the label being at a different protocol layer. In these cases, the policy should be the isolation of every data-unit known to that particular protocol function. As another example, some protocol functions have the capability to fragment and reassemble data-units. The reassembly security policy could be an exact-match, since it may not be appropriate for the protocol function to reassemble data-units using a dominance relationship even though the connected host supports such a policy.

CRITERIA OBJECTIVE: The capability to evaluate designs and implementations that support different security policies by component, protocol layer and protocol function.

3.4 External Security Policy

As discussed above, a uniform security policy is applied across the entire operating system. Communications systems may be required to adhere to several implementations of the DoD security policy mandated by the individual security requirements of the hosts they support. For example, the delivery criteria to a connected host may be an exact-match policy (data-unit sensitivity level must be at exactly the classification level of the host), rather than a dominance policy (data-unit sensitivity level must equal to or below the host classification level). As another example, some hosts have the authority to send data-units at a certain sensitivity level, but are not authorized to receive this level.

CRITERIA OBJECTIVE: The capability to evaluate designs and implementations where the system security policy is selectively applied to individual hosts attached to different parts of the system.

Operating systems lend themselves to a centralized reference monitor for mediating all access to data. This concept may not be appropriate for distributed communications systems where it it may not be acceptable to have all packets (or service requests) flow through a single component. The notion of distributed acceptance and delivery mediation with individual internal control mediation is more appropriate for communications system.

CRITERIA OBJECTIVE: The capability to evaluate designs that provide either centralized or distributed access control in the form of acceptance and delivery checks.

3.5 **Verification Assurance**

The criteria objectives of decomposition in terms of distributed components and protocol structures will only provide clear insights into the internal security properties of systems if there is means to verify these properties in a clear and consistent manner. The objective is to minimize the amount and types of verification evidence by limiting verification requirements to a small set that can be applied to any component or set of components identified in the decomposition process. Another objective is to reduce the amount of complexity by applying the criteria in a uniform manner in the decomposition process.

CRITERIA OBJECTIVE: The ability to apply the criteria recursively in the decomposition process, in terms of required security features and verification evidence.

3.6 **Why Not the Existing Criteria?**

The existing security evaluation criteria require excessive interpretation when applied to communications systems. The DoD Trusted Computer System Evaluation Criteria [3] (known as the Orangebook) are aimed at evaluating operating systems in the traditional sense. The application of these criteria to communications systems rely heavily on interpretation, since they were not specifically developed for evaluating systems with distributed components and/or hierarchical protocol structures.

In the future, operating systems will be evaluated against the Orangebook and their supporting communications systems will be evaluated against a communications criteria. To ensure consistent evaluation of the systems as a whole, there must be compatibility between the two criteria.

CRITERIA OBJECTIVE: Compatibility with the existing operating system criteria in terms of security features, verification evidence and assurance classes.

4. **Communications Criteria Requirements**

The criteria objectives can be integrated into a set of criteria requirements as follows:

a. Capability to evaluate communications systems based on a combination of distributed component and hierarchical protocol structures.

b. Capability to evaluate communications systems that implement different external(user-visible) and internal security policies.

c. Capability to evaluate communications systems that implement either centralized or distributed external security policies.

d. Capability to evaluate communications systems based upon well-defined and uniform design and verification evidence for both components and protocols.

e. Compatibility with the existing operating system criteria.

5. Proposed Criteria

Three concepts are central to the proposed criteria:

a. The criteria defines the decomposition approach to be taken and defines the level of decomposition based on the assurance class.

b. The criteria requires that the security policy be stated in terms on an Acceptance/Delivery Criteria and an Internal Control Criteria.

c. The criteria defines the structure, contents and level of detail for security features and verification evidence required for the various assurance classes.

5.1 Decomposition Approach

The specific decomposition requirements for communications systems is shown in Figure 1. For lower assurance classes, the decomposition approach begins at the level that identifies the Communications System, External Systems and associated interconnects. A Communications System Security Boundary is defined that encloses the Communications System. To be evaluated for a higher assurance class, the Communications System is to be decomposed into individual Nodes, and associated interconnects. A Node Security Boundary is defined for each Node. This first level decomposition supports the distributed nature of the Communications System.

To be evaluated at the highest assurance classes, the level of decomposition extends to the protocol layers, and the associated interconnect between these layers. This level of decomposition supports the layered protocol nature of communications systems.

5.2 Security Policy Definition

The Security Policy consists of an Acceptance/Delivery Criteria and an Internal Control Criteria. The Acceptance/Delivery Criteria determines how data-units are sent or received across the security boundary (Communications System, Node or Protocol). The Internal Control Criteria determines the relationship between data-unit while inside a particular security boundary. The two types of criteria must be complementary and, in combination, must be shown to implement correctly the stated Formal Policy Model.

5.3 **Documentation**

A uniform set of design documentation is used to support the evaluation process (see Figure 2). The set of documentation consists of design and verification documentation that is hierarchical starting at the top with the DoD Security Policy, and ending with a description of the underlying "virtual machine". Each piece of documentation produced is assumed to include the supporting evidence that it correctly represents the document above it. The documentation set has been structured such that it is additive, each higher assurance class requires additional documentation, but is supported by that documentation required by lower assurance classes. The documentation set is applicable to each component as it is identified in the decomposition process.

Appendix A is an example of the proposed decomposition and verification strategy.

5.4 **Summary of Assurance Classes**

The proposed criteria must be complementary to the existing criteria; therefore the communications system criteria closely parallels the existing criteria in terms of assurance classes.

The corresponding assurance classes of the two criteria is shown below.

Orangebook Criteria Assurance Class:	Communications System Criteria Assurance Class:
C1	X1
C2	X2
B1	Y1
B2	Y2
B3	Y3
A1	Z1

The communications system assurance classes are summarized as follows:

Assurance class X requires that data must be segregated in accordance with some stated security policy; however, no Acceptance/Delivery Criteria is imposed due to the lack of security labels. The principle difference between assurance class X1 and X2 is the amount of additional testing and documentation required.

Assurance class Y introduces decomposition in order to provide additional assurance. In this class data must be labeled.

Assurance class Y1 requires that a security policy and model must exist that accurately describes the communications system at the level seen by the External User. The security policy model must be stated in terms of an Acceptance/Delivery Criteria applied to the Communications System security boundary and an Internal Control Criteria that defines data segregation inside the Communications System.

Assurance class Y2, in addition to the Communications System security policy model, requires that a security policy model exist that accurately describes each physical node and link of the Communications System and must describe how the combination of these security policy models completely satisfies the Communications System Security Policy Model. The lower-level Security Policy Models are also stated in terms of an Acceptance/Delivery Criteria and Internal Control Criteria applicable to each node and link.

Assurance class Y3, in addition to the Communications System and Node/Link policy models, requires that a security policy model must exist that accurately describes each of the protocol layers implemented by the Communications System or Nodes. As above, these policy models must be shown to completely satisfy the upper level policy models. Criteria includes the use of existing specification and verification technology in the demonstration of the consistency among the policies, models and system components.

Assurance class Z requires no further mechanisms, but the level of assurance is raised by extensive use of formal methods, including formal specification and verification, the application of which is clearly identified and incorporated into the overall system development method.

6. **Issues**

A. Should the decomposition approach be directed by the criteria? That is, should the criteria force a designer/implementer in a specific direction that may skew the design or have other undesirable effects? The rationale basically is a desire for consistency; the decomposition requirement is central to the evaluation process and comparisons between two communications systems should be on an equal basis.

B. Should there be latitude for further decomposition?

C. Is the notion of an abstract TCB and supporting virtual machine sound? Is it helpful in the evaluation process?

D. Should the integrity of sensitivity labels in the protocol headers be modeled by formal methods? Clearly integrity issues are involved to the extent that the sensitivity labels in the various protocol headers must be protected from modification. Where sensitivity labels must be transferred from one protocol layer to

another the possibility of translation of such labels must be addressed.

E. What constitutes a valid protocol header? The format and content of a protocol header defines the location and (perhaps) the meaning of the sensitivity label. The validity of the sensitivity label therefore depends upon the validity of the format of the header.

F. What do we do about the higher protocol layers with additional functionality? This is where the Orangebook and proposed criteria meet.

G. Can the criteria be applied to commercial communications systems, where, perhaps, data corruption and authentication are more important than prevention of data compromise.

7. Conclusions

We believe there is sufficient justification for a security evaluation criteria for communications systems. The requirements for such criteria, as presented in this paper, are based on evaluation objectives suited to the characteristics of communications systems. The intent of the proposed criteria was to include as broad a range of such systems as possible. In that respect, the criteria have been used by Ford Aerospace as a basis of evaluation on current and proposed designs for several projects, including the Multinet Gateway, the WWMCCS Information System (WIS) and the Inter-Service/Agency Automated Message Processing Exchange (I-S/A AMPE). As communications systems, these projects represent a broad range of services and functionality. The criteria were found to be applicable to all three project designs. The proposed criteria require the incremental use of formal methods to provide increasing confidence in the validity of the system security properties. The rigor imposed by these methods is considered essential in evaluating and certifying complex communications systems.

8. Acknowledgements

The author wish to thank A. Paul Cook, George W. Dinolt, James W. Freeman and Richard B. Neely for their helpful and perceptive comments.

9. References

[1] Proposed Draft Recommendation X.200, Reference Model of Open Systems Interconnection for CCITT Application, CCITT SG VII/WP5, Special Rapporteur of Layered Models, December 1982.

[2] DoD Protocol Reference Model, TM-7172/201/02, System Development Corporation, January 1983.

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

Figure 1 - Decomposition Requirements

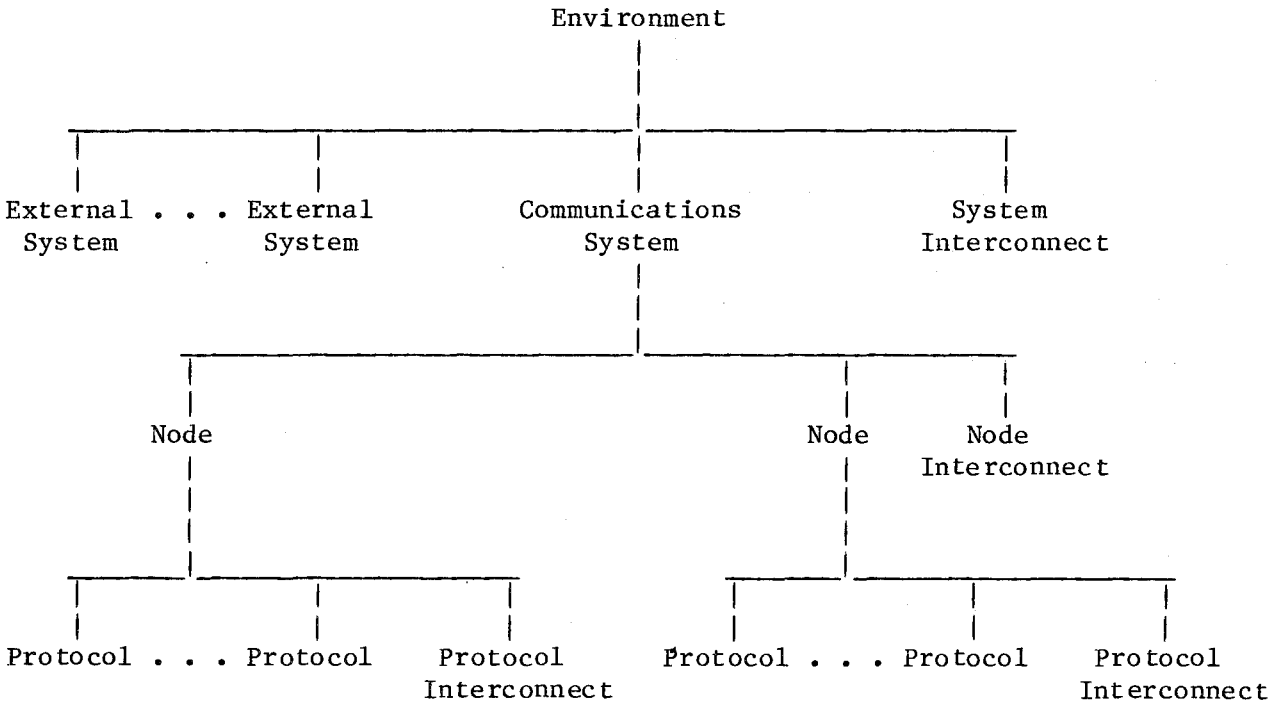
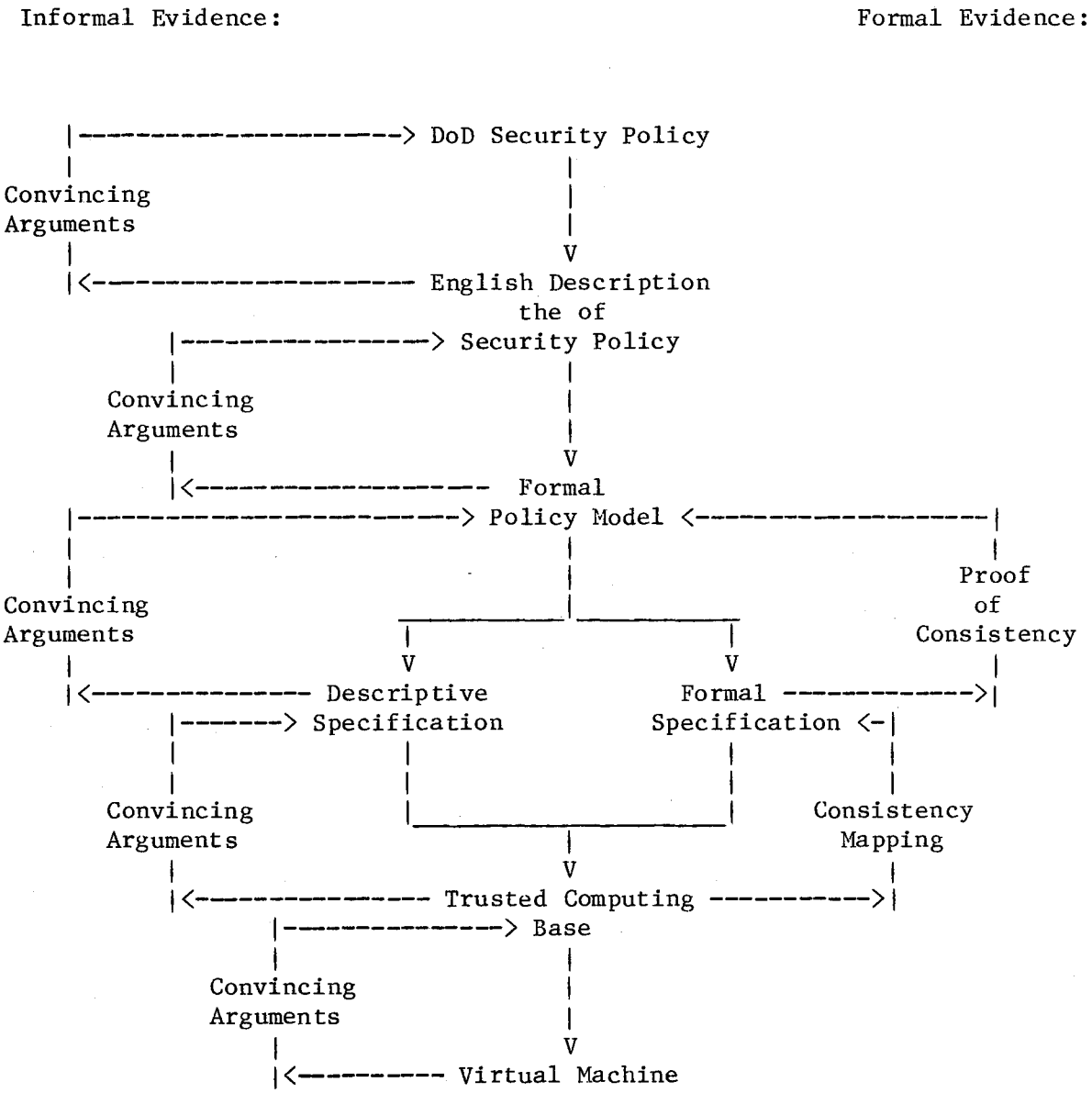


Figure 2 - Communications Criteria Documentation Tree



A. The System (s-Component) is covered first:

1. The s-English Description of the Security Policy is developed and is shown by convincing arguments to satisfy the DoD Security Policy.

2. The s-Formal Policy Model is developed in terms of the s-Acceptance/Delivery Criteria and s-Internal Control Criteria and is shown by convincing arguments to satisfy the s-English Description of the Security Policy. The s-Formal Policy Model is proven internally consistent.

3. The s-Descriptive/Formal Specification is developed and is shown by convincing arguments and/or proof of consistency to satisfy the s-Formal Policy Model.

4. The s-TCB is developed and is shown by convincing arguments and/or proof of consistency to satisfy the s-Descriptive/Formal Specification. (The s-TCB is probably abstract.)

B. The Node (n-Components) are covered next:

1. The s-Component is decomposed into one or more n-Components.

2. An n-English Description of the Security Policy is developed for each n-Component and they are in combination shown to satisfy the s-English Description of the Security Policy.

3. An n-Formal Policy Model is developed for each n-component in terms of the n-Acceptance/Delivery Criteria and n-Internal Control Criteria and is shown by convincing arguments to satisfy the respective n-English Description of the Security Policy. Each n-Formal Policy Model is proven internally consistent. The combination on n-Formal Policy Models are shown to satisfy the s-Formal Policy Model.

4. An n-Descriptive/Formal Specification is developed for each n-Component and is shown by convincing arguments and/or proof of consistency to satisfy the respective n-Formal Policy Model. The combination of n-Descriptive/Formal Specifications are shown to satisfy the s-Descriptive/Formal Specification.

5. An n-TCB is developed for each n-Component and is shown by convincing arguments and/or proof of consistency to satisfy the respective n-Descriptive/Formal Specification. The combination of n-TCBs are shown to satisfy the s-TCB. (The n-TCB may be abstract.)

C. The Protocol (p-Components) are covered next:

1. Each n-Component is decomposed into one or more p-Components.

2. A p-English Description of the Security Policy is developed for each p-Component and they are in combination shown to satisfy the n-English Description of the Security Policy.

3. A p-Formal Policy Model is developed for each p-component in terms of the p-Acceptance/Delivery Criteria and p-Internal Control Criteria and is shown by convincing arguments to satisfy the respective p-English Description of the Security Policy. Each p-Formal Policy Model is proven internally consistent. The combination of p-Formal Policy Models are shown to satisfy the n-Formal Policy Model.

4. A p-Descriptive/Formal Specification is developed for each p-Component and is shown by convincing arguments and/or proof of consistency to satisfy the respective p-Formal Policy Model. The combination of p-Descriptive/Formal Specifications are shown to satisfy the n-Descriptive/Formal Specification.

5. An p-TCB is developed for each p-Component and is shown by convincing arguments and/or proof of consistency to satisfy the respective p-Descriptive/Formal Specification. The combination of p-TCBs are shown to satisfy the n-TCB.

SECURITY ISSUES INVOLVED IN NETWORKING PERSONAL COMPUTERS

Alfred Arsenault

DoD Computer Security Center

INTRODUCTION

In recent months, there have been several preliminary attempts made at writing trusted computer network evaluation criteria, similar to the standards established in the Department of Defense Trusted Computer System Evaluation Criteria. One of the goals of a trusted network evaluation criteria should be to be as widely applicable as possible. If possible, it should apply to both long haul networks, such as ARPANET, and local area networks, or LANS. Also, it should apply to cases in which hosts attached to the network are mainframes, as well as when hosts are microprocessor-based personal computers, or PCs.

A major difficulty encountered in writing trusted network evaluation criteria is that the definition of a "secure network" is not fully agreed upon. In attempting to define what is meant by a "secure network", it is necessary to consider the issues involved in network security. It has been discovered that there are several cases in which a security issue relates to only one particular type of network. One of the types of networks which has a large number of unique problems is the case in which several personal computers in a close geographical area are linked together, possibly with other devices, to form a local area network. It is on this special case that this paper will concentrate.

CAUSES OF SECURITY PROBLEMS

There are two basic causes for most of the security problems that arise specifically because most of the hosts on a LAN are PCs: first, because a PC is a single state machine, and second, because where PCs are concerned, communication protocols have traditionally been weak from a security standpoint, with few or no security features designed in.

Problems Caused by Single State Machines

First, consider the problems caused by the fact that PCs are single state machines. A single state machine has no dominance domain. A dominance domain, also called a supervisory state, is a characteristic of most large main frame computers that restricts the ability of certain users to access certain locations in the machine's memory. Thus, a PC has no capability to support a Trusted Computing Base, or TCB, because no hardware/software security mechanisms can be tamperproof. For this reason, any single state machine is a division D system, according to the Department of Defense Trusted Computer System Evaluation Criteria.

Currently, in many LANs having PCs as hosts, many crucial network functions are implemented in the PC itself, and, since the PC has no dominance domain, there is no way to prevent a penetrator from accessing those network functions and changing them to allow violations of the network security policy. Thus, a penetrator can evade any security mechanism that is implemented in the PC. This is clearly not desirable in the operation of a secure network.

Problems Caused By the Network

There are two characteristics of most LANs that account for many security problems. First, particularly where PCs are concerned, communication protocols historically have tended to be very weak from a security standpoint, with few or no security features designed in. Second, the transmission method in many LANs is broadcast.

From a security point of view, the protocols are among the most important functions of the network. The problem is that most PC communication protocols are very simple in comparison to those used in mainframes. They are usually designed only to insure that a message sent from one host to another is transmitted properly, and, upon arrival at the destination host, is received properly. From a purely operational or functional standpoint this is sufficient in most cases, but from a security point of view it is less than desired. Well designed, security-oriented protocols tend to be slow, and to take up much memory space. This has resulted in their being unfeasible for use in many microprocessor-based PCs, which have had slow processing speeds and small memories. The situation should be improving, however, because PCs now have much larger memories and higher speeds than they used to. This development improves the capability of the PC to handle sophisticated protocols, with much more security designed in.

LANs having PCs as hosts using broadcast transmission is a second cause of many major security problems. This is true regardless of whether the actual communications medium is twisted pair wire, coaxial cable, optical fiber, or even satellite transponders. Broadcast transmission can lead to spoofing, wiretapping, and the surreptious entry of messages into the communications medium in an attempt to violate the security of the network. These problems appear to exist on any LAN using broadcast communications, regardless of whether ring, bus, or some other architecture is used.

ISSUES AND PROBLEMS: SOME EXAMPLES

Consider now some of the actual security issues and problems that arise from the networking of computers. There are four areas in which the interconnection of PCs to form a LAN causes problems that either do not exist in other types of computer networks or are significantly different in the case under consideration. These areas are: access controls, spoofing, wiretapping, and auditing. Each of these topics will be discussed in turn.

Access Controls

Access controls are one of the principal mechanisms used to prevent the compromise of data. Access controls are used to restrict the ability of users to read from or write to a specific memory location. Access controls are vital to the security of a network; however, there is no way to implement either mandatory or discretionary access controls in a single state machine, since any user with access to a single state machine can access any memory location known to the machine, regardless of what is stored there.

This lack of access control extends not only to the host being directly accessed by a penetrator, but also to any host which can be remotely accessed. That is, if a user can access another PC over the network, that user can access any memory location known to that remote PC, usually including anything stored on any disks located in the system's disk drives at that time.

On the type of network under consideration, access controls cannot be used to prevent a penetrator from accessing network functions that are implemented in the PC itself. As an example, consider a LAN using a contention scheme or collision detection algorithm. A penetrator can subvert the contention algorithm on his host, and send out a continuous stream of messages. This will deny use of the network resources to other, legitimate users, some of whom may have urgent messages to send. The worst part of the scenario discussed here is that it may be impossible for other network users to determine exactly what has happened.

Spoofing

Another common network security problem is spoofing. Spoofing occurs when one user pretends to be another user, an operating system, or even the network, in an attempt to cause a second user to violate the security of the network. Spoofing is usually relatively easy to do in the case under consideration, and is very difficult to detect.

Spoofing plays a part in data integrity violations. Suppose that "host #22" sends a message to "host #5", but labels the message as if it came from "host #19". This is a spoof, because "host #22" is pretending to be something it is not; namely, "host #19". In the type of network under consideration, it is likely that the recipient of the message, "host #5", will not be able to detect the spoof. "Host #5" may then proceed to perform some action for which "host #22" has no authorization to request, but for which "host #19" does.

Wiretapping

Although wiretapping can occur on all networks, the fact that the transmission method is broadcast makes it an even more serious problem than usual. Each host on the network sees all of the messages being transmitted across the network, instead of a fraction of them, as is the case in a packet switched network. This greatly increases the damage that can be done by one penetrator: he can cause the compromise of all data sent on the network, instead of a fraction.

For example, suppose that a message is sent by one host, and is addressed to "host #22". While the message is passing along the network, "host #19" copies the message off of the network. ("Host #19" must at least look at the header of the message to determine whether or not is is the intended recipient.) Since all hosts are essentially taps on the communications medium, it is quite likely that the fact that the message has been copied will not be detected. This is a compromise of data.

Auditing

Consider now the problem of auditing a LAN composed primarily of PCs. There are two basic strategies for auditing a network: auditing the actions of each node at that node itself, or auditing the traffic flow at a small number (possibly one) of centralized locations. If auditing were implemented such that each node kept an audit trail of its own actions, a user could then alter the audit trail of his actions. A penetrator could then incur numerous security violations, and erase any record of their having occurred. Therefore, any audit trail obtained for the network from the nodes themselves would be essentially worthless, since it would contain records of only those actions which the users wanted others to know about.

The alternative scheme would be for auditing to take place at a small number of centralized locations. However, in this scheme, the audit mechanism could not detect much of the surreptitious copying of messages taking place. That is, it would not be able to detect a large number of the security violations that could be occurring throughout the network. Since it is important that some type of audit trail be kept of the activities occurring on the network, detecting all security violations is an important objective, and auditing is a very difficult problem.

POSSIBLE SOLUTIONS

Now that several of the security problems associated with connecting PCs in LANs have been outlined, consider several possible solutions, along with some of the advantages and drawbacks of each.

"Network High" Mode

One solution currently in common use is to keep all PCs connected to the LAN operating at the same "network high" level, and only allow communication at a single sensitivity level. That is, all hosts attached to the network can be physically accessed only by people who are trusted to at least the "network high" sensitivity level, and all communications over the network are considered to be at that level. This is analogous to the system high operating mode of stand-alone computer systems. This strategy results in letting only trusted users have access to the LAN, and in basing our trust in the network users rather than the hardware/software mechanisms. While this method is more secure than simply allowing unrestricted access and communication over a range of levels, it can lead to instances of multiple LANs when communications are necessary at more than one level. This is generally less efficient and more costly than open communications. It should be remembered that the primary purpose of a LAN is to facilitate communication between users, and too many restrictions on communication would defeat this purpose.

Better Protocols

A second possible solution is to design a "better" set of protocols that would allow multilevel communication; i.e., would allow messages to be sent at one of several distinct sensitivity levels. This would allow wider communication among users, and would lead to greater efficiency. For example, good access controls can be designed into communications protocols. A protocol could contain the mechanisms necessary to support mandatory access controls. Therefore, all data coming from a host would be appropriately labeled, as would all access requests arriving at that host. Any access request arriving without an appropriate sensitivity label would be refused. One major problem with this approach is that well designed, security-oriented protocols that allow for labelling, access control decisions, and auditing tend to be very slow and take up a great deal of memory. This may be overcome shortly, due to the rapidly expanding memory sizes and processing speeds of newer PCs. However, as long as these protocols are implemented in the PCs themselves, they are susceptible to subversion, and thus do not completely solve the problem.

Encryption

A third possible solution is the use of encryption. All messages sent between hosts can be encrypted with a suitable encryption algorithm. This can help solve the data compromise problem - a spoofer may still be able to copy messages without being detected, but may not be able to decipher and understand the stolen information. Encryption might also help solve the data integrity problem since messages which have been changed during transmission can be detected, and a good encryption scheme would help validate the authenticity of return addresses on messages received. However, encryption will do nothing to help prevent certain types of integrity or denial of service problems. A penetrator can still inject an endless stream of messages onto the transmission medium. Encrypting the message does not prevent this, and in fact has no effect on the entire situation.

Trusted Interface Devices

Another solution would be the use of a trusted interface device, or front end, between the PC and the communications medium. This could definitely help solve the data compromise problem, because the front end could prevent a PC from spoofing the system by only giving the system messages actually intended for it. It would also help solve the data integrity problem if it attached the message headers itself and was trusted to attach only the proper ones. A trusted front end may also provide help in solving the denial of service problem. For example, if the network used a contention scheme, and the front end enforced the backoff time algorithm and could not be reprogrammed by the user, the penetrator could no longer send an endless stream of messages onto the network. The denial of service problem could be lessened in that respect.

The key factor in this solution would be the fact that the user could NOT reprogram the trusted front end to subvert its security features. This method, alone or in combination with some of the others previously mentioned, is probably the best hardware/software solution to the problem of LAN security currently under consideration. It is, however, one of the most expensive, since each host on the network would require a complete (hardware and software) front end device, and these would not be cheap to develop in such a manner that they would be considered to be trusted.

Current Best Strategies

The best security strategies currently in use are administrative, physical, and "common sense" strategies. These include letting only "trusted" users have access to the network, keeping disks locked up when not in use and out of the machine when not necessary, and paying close attention to the classification and compartmentation of messages sent and received over the net. Until one or a combination of the above solutions is fully implemented, these strategies will continue to be the best available.

The Euclid Family and its Relation to Secure Systems

Glenn H. MacEwen
David T. Barnard

Andyne Computing Limited
Kingston, Ontario

and

Department of Computing and Information Science
Queen's University, Kingston, Ontario

Abstract

This paper discusses the evolution of, and the technical differences between, the various versions of the Euclid programming language that have appeared since the original publication of a language designed for verfiable systems programs. In addition, some current work directed at transporting verified Euclid programs into Ada systems is described. The motivation for this transporting work is to provide trusted software within Ada systems.

Introduction

The language Euclid was first developed for use in secure operating systems. However, it never was used for that purpose and since its original introduction it has evolved through several variants. As Euclid was evolving, the Ada language has appeared as a military standard raising the question as to the relation of secure Euclid systems to Ada environments.

This paper reviews the current variants of Euclid very briefly and then looks at the question of the Euclid/Ada relationship. The approach of transporting Euclid programs into Ada environments is discussed in the context of two experimental projects. Finally, some observations regarding the requirements for concurrency support in Euclid and its associated verifiers are made.

Euclid

The original definition of Euclid was published in 1977 [Lam77, Lon78, Pop77]. That report states that the language is intended for the expression of system programs that are to be verified:

> "By a ***verifiable*** program we mean one written in such a way that existing formal techniques for proving certain properties of programs can be readily applied; the proofs might be either manual or automatic, and we believe that similar considerations apply in both cases. By ***system*** we mean that the programs of interest are part of the basic software of the machine on which they run; such a program might be an operating system kernel, the core of a data base management system, or a compiler."

The language explicitly is not a general-purpose one, and does not address the problems of constructing very large programs.

Euclid is based on Pascal. The main changes are restrictions, thus allowing stronger statements about the properties of a program to be made based on compiler analysis. The main differences from Pascal include:

There is explicit control of identifier visibility (via import and export specifications in programs) rather than implicit inheritance of containing scope.

Aliasing (referring to the same or overlapping variables by two identifiers in the same scope) is prohibited.

Pointer variables are constrained to refer to an object in a specified collection.

Dynamic allocation is under program control, but is constrained.

A type can have a formal parameter allowing arrays with bounds fixed at creation, and type-safe variant records.

Modules are an encapsulation mechanism for variables, types, and routines.

The concept of a constant is extended to include a variable whose value cannot change in the present scope.

A module can be a generator, producing values to be used in a for statement.

There are explicit mechanisms to access the underlying machine, and to override type checking.

Assertions can be inserted in programs, and code to check them can be generated.

Some features present in Pascal are omitted, including input and output, real numbers, multi-dimensional arrays, (arrays of arrays are permitted) labels and gotos, and functions and procedures as parameters.

Other considerations in the design were the need to run programs on several machines (thus several code generators would be required), the need for efficiency of generated code without excessive compilation cost, and the need to keep run-time support to a minimum so that verification could be done.

In addition, the language design was intended to be based on current knowledge of programming languages and compilers, and thus not include features not understood or difficult to implement. This, as later developments showed, was not achieved. In particular, the first completed implementation (see the following section) uncovered several problems. A revised language definition, call Full Euclid (FE), was published in 1981 [Lam81].

Toronto Euclid

The first successful implementation of Euclid was a joint effort of I.P. Sharp Associates and the University of Toronto [Hol78a, Hol80, Wor81a, Wor81b, Pas80]. The work to produce this compiler resulted in a version of the language named Toronto Euclid (TE), and pointed out a number of problems with the original definition.

TE is best characterized as a subset of FE, although there are a number of minor extensions or modifications to the language. For example, machine code routines are Unix assembly code (for the PDP-11); this feature is not part of FE. There are some restrictions such as requiring expressions in assertions to be enclosed in parentheses, and ignoring case in the spelling of identifiers.

Some features of the FE language are not included in TE. For example, finalization routines for modules are not allowed, there is no inline expansion of routines, types returned by functions must be scalar or set types, there are no parameterized types, there is no range checking on assignment, separate compilation is not supported, and variant records are not allowed. Most of these are relatively minor as far as security is concerned.

A relatively successful research project to translate TE programs into Ada was carried out between 1982 and 1983 by Andyne Computing [Lee84]. The rationale for this investigation is that critical software, such as trusted secure components, can be verified in Euclid and then transported to an Ada environment in which they are required to run due to language standardization. The important issue of the security of the Ada run-time system was not addressed. This TE/Ada translator, built by modifying the TE compiler, is able to translate a significant subset of TE. We call this subset Kingston Euclid (KE).

Although an axiomatization for Euclid was produced [Lon78] (see also [Cra82b]), no work on support for verification of TE programs has been done.

Concurrent Euclid

Concurrent Euclid (CE), was developed by the team at the Computer Systems Research Group at the University of Toronto that had been involved with the Toronto Euclid project [Cor81, Hol83]. CE can best be explained as a subset of FE, together with a set of extensions based on monitors to support concurrency.

The sequential subset is very similar to the TE variant of the language. One area of difference is in passing parameters to routines. CE allows the upper bound of an array index type to be the word ***parameter*** thus signifying that any array with the appropriate element type and an index type with the appropriate lower bound can be passed. CE also allows the type of a parameter to be ***universal*** thus signifying that any type can be passed, and that the object will be viewed as an array of StorageUnits.

The concurrency features of CE are based on monitors as described by C.A.R. Hoare [Hoa74]. Each ***module*** can contain any number of processes. A ***process*** is declared like a parameterless procedure, and begins execution following the initialization of the module. Processes can alter global variables. The approved way for processes to communicate is via monitors; a ***monitor*** is a special kind of module. A monitor exports procedures and functions, which are called entries. The implementation guarantees that at most one process is executing inside a monitor--i.e., is executing one of the monitor's entries--at any

time. Thus, a process is guaranteed exclusive access to the monitor's variables when in an entry, since a monitor cannot export variables. A process attempting to enter the monitor when another process is already executing inside the monitor, is blocked.

Variables of type ***condition*** are provided, along with the operations ***wait*** and ***signal*** so that processes can explicitly block themselves until some logical condition obtains, and can send notification to other processes that the condition on which they are waiting has been established.

There is also a ***busy*** statement so that a simulated time can be advanced. Programs run in zero elapsed simulated time. CE also provides support for separate compilations (via the ***external*** attribute of a unit) and linking of separately compiled units.

A project is currently underway at Andyne Computing to provide a mechanism to move compiled CE programs into an Ada run-time environment. This provides an alternative to the translation method used in the TE/Ada project.

We are currently able to produce, using a modified CE compiler, a load module that can be linked and run with Ada programs. (The system in use in an Intellimac 68000-based machine running Telesoft Ada under Unix.) The language interface that we expect to support is a CE module/Ada package with exported procedures and functions having only primitive typed parameters: integers, booleans, characters, and strings. We also expect, in follow-on work, to provide internal concurrency by building on the Ada run-time kernel. That is, the CE module will probably be a monitor providing mutual exclusion for its operations. We say more about this issue of concurrency later.

Some preliminary work has been done toward an axiomatization of CE and the construction of a verification condition generator (VCG) [Mat82]. However, concurrency was not addressed in this work and it has not been continued beyond the initial prototype. We are not aware of any effort to pursue support for verification of CE programs.

Ottawa Euclid

During the development of CE, I.P. Sharp Associates proceeded in a different direction that resulted in Ottawa Euclid (OE), which can be described as FE with extensions to support verification [Cro81,Cro82,Cra83a]. In particular, an ***external module*** mechanism for specification and a ***theory*** facility were added [Cra83b].

The unit of specification in OE is the Euclid module, represented by an external module declaration containing routine specifications, invariant specification assertions, and variable and constant declarations. Functions and constants for proof purposes only can also be declared.

The theory construct provides linguistic support for the encapsulation of reusable definitions of mathematical objects and their properties. The need for theories follows from the necessity to provide, in a program verification, axioms, lemmas, and theorems about the domain of computation of the program under scrutiny. For example, a program manipulating tree data structures requires definitions of the meaning of tree operations. Since tree manipulations are common in many programs it is clearly desirable to define the required tree properties once and for all in a way that is useful in many different programs. OE theories provide a way to define such domain specific knowledge so that it is reusable across many programs.

A named theory can be included in a program and referenced in a similar way as is a Euclid module. Details of the language of external modules and theories are described in [Cra83b].

A concurrency mechanism was not considered for inclusion among the OE extensions for two reasons. First, it was judged that the task of formalizing Euclid satisfactorily was sufficiently difficult that to attempt to incorporate concurrency would significantly

reduce the chances of success. Second, it was not clear what mechanism would be appropriate and that there was a good chance that the mechanism chosen could very well end up simply interfering with the specialized kinds of software that were envisioned as typical for OE applications.

The OE compiler, which is based on the TE compiler structure except for the replacement of a more machine-independent code generator [Lan82], is expected to be completed during 1984.

Although the extensions from FE to OE are intended for verification, it is not intended that full OE be supported with verification tools. Consequently, I.P.Sharp is currently completing the definition of a set theoretic model to express formally the semantics of a subset of OE which will be supported.

Verifiable Ottawa Euclid

I.P Sharp is currently finalizing the definition of Verifiable Ottawa Euclid (VOE), a subset of OE suitable for verification [Cra83c,Cra82b]. A set theoretic mathematical formalism based in part on the notion of state relations is being developed by I.P. Sharp to define the semantics of VOE. A verification logic and a VCG will be produced for the analysis of OE programs. It then will be possible to prove the soundness of the logic and the VCG with respect to the semantic definition of VOE.

VOE is to be a part of the Euclid-based Verification and Evaluation System (EVES) which will provide, in addition to VOE, a theorem prover, and a variety of programming support tools [Bon82,Cra84]. The VOE subset is expected to be completely defined by the fall of 1984; the statement semantics are described in [Cra83c] but expressions and, particularly, data types have not yet been finalized.

An initial application of EVES is to be the formal verification of the LSI Guard [Cra82a] which has been implemented in TE and specified in OE. The implementation incorporates a simple run-time kernel with primitives for voluntary context switching of processes.

Some Observations

The two Euclid/Ada transporter projects have been based on different variants of Euclid, TE and CE. Both projects have been successful in the sense that in each we have accomplished what we set out to do with less difficulty than we were prepared for. It is now clear that if this approach is to be useful and to be carried further we must consider what Euclid is appropriate for further development. To do this, we must identify potential target applications and carefully consider the requirements for them. This will accomplish two things: (a) validate the need for such a mechanism, (b) help in making the language decision based on these requirements.

After considering several potential circumstances where a verified Euclid module could be useful, it has been concluded that most useful examples fall squarely into the **reference monitor** abstraction. Any application which involves secure access to shared resources falls into this category. Isolation of subjects (processes) and concurrent access to resources are inherent.

Another potentially useful, and simpler, model is the verified procedure. This is simply a sequential procedure, linked with an application program, which performs some critical computation such as the authentication of a password or a critical user command. This seems much less useful, however, since the correctness of the calling Ada code is of equal importance to the procedure.

Figure 1 illustrates a plausible structure for the design of a reference monitor written in Euclid but running in an Ada environment.

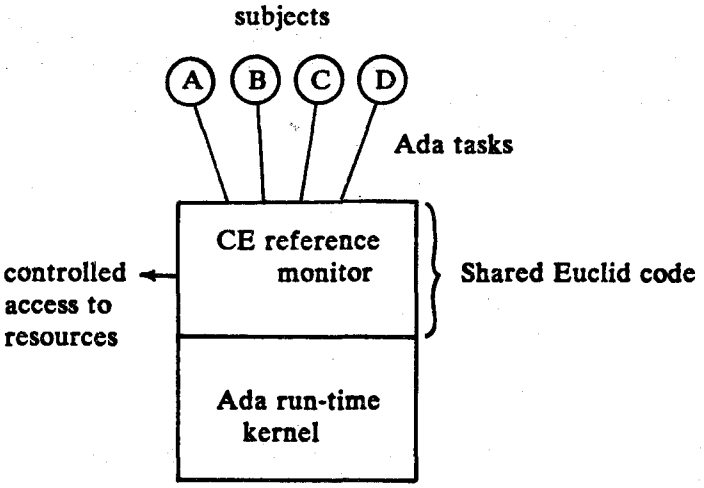


Figure 1 A Euclid Reference Monitor

A number of Ada tasks are shown running on the Ada kernel. Each task has been linked with a Euclid reference monitor (RM). For efficiency, it is assumed that only one copy of the RM is used so that is implied in the diagram. Conceptually, however, each task has its own copy of the RM. The procedure call interface is used because that is what is provided by the CE/Ada system. This structure has some inherent assumptions that are significant.

Assumption 1: An Ada program and the RM can be linked in a secure way.

This assumption is necessary to ensure the isolation of the tasks from each other and from the resources. The only interaction of a task with its environment must be via RM or the Ada kernel.

Assumption 2: The Ada run-time kernel is secure.

This assumption is necessary to ensure that the RM has exclusive control over and access to the resources. It is also necessary for the isolation of tasks.

Assumption 3: RM has a secure way to authenticate the identity of calling subjects.

For example, certain of the resources may be subject terminals which produce an authentication (password) message and are then associated with a task.

All of these assumptions require more detailed analysis to define precisely what "secure" means in each case. Our intent here is not to do that, but to identify the implications of this structure on the Euclid/Ada transporter mechanisms.

The major implication is that such an RM is a shared concurrent program. Consequently, a minimum requirement is for some form of mutual exclusion. Since the RM is to be verified then this, in time, places requirements on the verification system.

The most direct approach is to use CE monitors. The CE/Ada system would need only to use a semaphore-like facility from the Ada kernel to support the monitor concurrency mechanism. The verification system must, of course, be capable of handling monitors.

The final conclusion, then, is that most useful applications will require concurrency and so this should be addressed in the design of the Euclid/Ada system as well as in the design of a suitable Euclid verification system.

Acknowledgement

Dan Craigen of I.P. Sharp read a draft of this paper and made many helpful suggestions and clarifications.

References

[Bon82]
Bonyun, D., et al.
A Blueprint for a Verification and Evaluation Environment based upon Euclid
IPSA Technical Report FR-5017-82-1, 1982.

[Cor81]
Cordy, J.R., Holt, R.C.
Specification of Concurrent Euclid
Computer Systems Research Group
University of Toronto
Technical Report CSRG-133, August, 1981

[Cra82a]
Craigen, D.
A Formal Specification of the LSI Guard
IPSA Technical Report TR-5031-82-2, August 1982.

[Cra82b]
Craigen, D.
The Euclid Proof Rules: A Critique
IPSA Technical Report FR-5091-82-1, September 1982.

[Cra83a]
Craigen, D.
Towards a Formal Semantics for Ottawa Euclid
IPSA Technical Report FR-5140-83-1, August 1983.

[Cra83b]
Craigen, D.
The Theory Construct in Ottawa Euclid
IPSA Technical Report FR-5146-83-1, September 1983.

[Cra83c]
Craigen, D., Saaltink, M.
A Formal Semantics of VOE: Part 2
IPSA Technical Report FR-5172-83-1, November 1983.

[Cra84]
Craigen, D.
Ottawa Euclid and EVES: A Status Report
IEEE Symposium on Security and Privacy
Oakland, Ca, April, 1984.

[Cro81]
Crowe, D. R.
Ottawa Euclid language specification
IPSA Technical Report TR-5613-81-7, November, 1981.

[Cro82]
Crowe, D.
Ottawa Euclid Reference Manual
IPSA Technical Report TR-5163-81-7, December 1982.

[Hoa74]
Hoare, C.A.R.
Monitors: an operating system structuring concept
CACM 17, 10(Oct 74), 549-557.

[Hol78a]
Holt, R., et al
The Euclid Language: A Progress Report
Proceedings of ACM National Conference, December 1978.

[Hol80]
Holt, R.C., et al
The Toronto Euclid Compiler Project Workbook
IPSA Technical Report, March 1980.

[Hol83]
Holt, R.C.
Concurrent Euclid, The Unix System, and Tunis
Addison-Wesley, 1983.

[Lam77]
Lampson, B.W., et al
Report on the Programming Language Euclid
ACM SIGPLAN Notices 12, 1(Feb 77).

[Lam81]
Lampson, B.W., et al
Report on the Programming Language Euclid
XEROX Parc Technical Report, October 1981.

[Lan82]
Landwehr, R., et al
Experience with an Automatic Code Generator Generator
SIGPLAN Symposium on Compiler Construction
June, 1982.

[Lee84]
Leeson, D.A., Barnard, D.T., MacEwen, G.H.
Issues and Experience in Building a Euclid-to-Ada Translator
in preparation.

[Lon78]
London, R., et al
Proof Rules for the Programming Language Euclid
Acta Informatica 10, 1978.

[Pas80]
Pase, B.
Toronto Euclid Language Specification
IPSA Technical Report TR-3819-80-2, January 1980.

[Pop77]
Popek, G.J., Horning, J.J., Lampson, B.W., Mitchell, J.G., London, R.L.
Notes on the design of Euclid
Proceedings of ACM Conference on Language Design for Reliable Software
SIGPLAN Notices 12,3 (March 77), 11-18.

[Wor81a]
Wortman, D., et al
Euclid - A Language for Compiling Quality Software
National Computer Conference, Chicago, May 1981.

[Wor81b]
Wortman, D., Cordy, J.
Early Experiences with Euclid
5th International Conference on Software Engineering, March 1981.

A COMPARISON OF FORMAL SECURITY POLICY MODELS

J.T. Haigh
Martin Marietta Aerospace
Denver, Colorado 80201

Abstract

This paper complements the work of Taylor presented at the 1984 IEEE Symposium on Security and Privacy. It presents a formal analysis of the similarities and differences among three models and policies for multi-level security (MLS), the Bell and LaPadula Model, the SRI model developed by Feiertag, Levitt, and Robinson, and the non-interference model developed by Goguen and Meseguer. The major difficulty encountered is in translating one model and policy into the language of the model with which it is to be compared. The main result is that the Bell and LaPadula policy is the most restrictive of the three, while the Goguen and Meseguer policy is the least restrictive. Under certain conditions the three policies are equivalent.

FOREWORD

This report was prepared by Martin Marietta Denver Aerospace. The effort was conducted in the Computer System Engineering Section of Systems Engineering, Space and Electronic Systems Division under Project Authorization D-72R of calendar year 1983.

I. Introduction

In the DoD Computer Security Center's "Trusted Computer System Evaluation Criteria" the requirements for certification at the B2 or higher level include the requirement that a formal security policy for the system be maintained and arguments be given which show the top level specification for the system is compatible with the formal security policy. These arguments comprise what is known as the verification of the top level specification with respect to the security policy. In this paper three different formal models and policies for multi-level security (MLS), based on informal natural language security policies which differ in their control objectives, are examined. The models and policies are those due to Bell and LaPadula [1], Feiertag, Levitt, and Robinson [4], and Goguen and Meseguer [5] respectively. A natural question to ask is: Are these policies equivalent? That is are they really the same policy in disguise? If they are not, what are the differences among them? If they are, what are the desirable and undesirable features of each of the formulations? The first published study of these questions is that of Taylor [15]. The main result of this paper is that the three policies are, in general, different. The first policy is most restrictive and the third is least restrictive. However there are circumstances in which the policies are equivalent.

Goguen and Meseguer distinguish between a formal security policy and a model for the system to which the policy should apply. This distinction is maintained here. The security policy is formally described in terms of the elements of the model, and normally an attempt is made to prove the model satisfies the policy. One of the difficulties in comparing the security policies is that each is formulated in terms of a different model of the system. It is necessary to translate a policy formulated in terms of one model into a formulation in terms of another model before the policies can be compared. In an abstract setting the interpretation is somewhat subjective since there is no way to prove an interpretation is correct except in the context of a particular system, when the mappings from the system to each model can be compared. In a very nice series of draft reports, Rushby [10, 11, 12] develops a set of assumptions for the Bell and LaPadula policy which he uses to show it implies an instantiation of the Goguen and Meseguer policy. Similiar assumptions are made, either implicitly or explicitly, in this paper. These will be indicated in the text.

Each of the models refers to a lattice, K, of security levels. In order to orient the reader, each security level will be thought of as a DOD classification level, unclassified to top secret, along with a set of categories of information. The security level of a user, u, will be denoted by K(u). It indicates u has some sort of rights to information at certain classification levels, contained in the set of categories of K(u). However it is important to remember some other lattice could be used. In particular the lattice could be an integrity level lattice as described by Biba [2] or a more comprehensive protection level lattice incorporating both security and integrity levels. The paper by Grohn [7] develops the notion of protection level very nicely. The formal policies and the results discussed in this report apply to any lattice of "security" levels. Each of the policies and the differences among them will be described informally in the body of this report. The appendix contains formal descriptions of the models as well as proofs of the relationships among them. While the proofs themselves are quite simple, the descriptions of the models do require extensive notation.

II. The Bell-LaPadula Policy

The Bell and LaPadula policy [1] is based on the notion of access control. It is essentially a formalization of the DOD paper and file cabinet policy adapted to an automated environment. The primitive elements of the model are:

1) Subjects: the active entities of the system, including users and processes acting on behalf of users;

2) Objects: the passive entities of the systems, the high-level data structures in which information is stored;

3) Observe, modify: the modes of access which subjects may have to objects, denoted $\underline{o}$ and $\underline{m}$ respectively; and

4) K: the lattice of security levels for the subjects and objects.

These elements and relations on them are used to define the state of the system. For this report the crucial component of the state is the current access list which consists of the set of all triples of the form (subject, access mode, object) such that the subject can currently access the object in the given mode. The multi-level security policy, denoted BL, consists of two properties of the current access list, the simple security and * (pronounced star) properties, as well as three properties which essentially keep the security levels of objects constant from their creation to their deletion and the levels independent from one creation to the next. For this paper BL is:

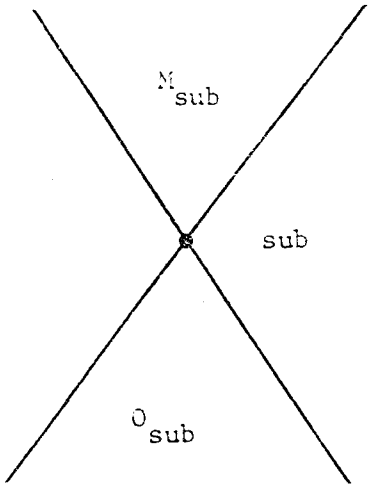
Definition: A state is multi-level secure if for each triple (subject, x, object) in the current access list, the following properties are satisfied.

ss) The level of subject is at least the level of object whenever x = o.

*) The level of object is at least the level of subject whenever x = m.

T) During each incarnation of a subject or object, its level remains constant.

The conditions expressed by (ss) and (*) are illustrated in Figure 1. If sub is any subject, then O_{sub} and M_{sub} are the cones of objects which are respectively observable and modifiable by sub. Since the set K is only partially ordered, there will presumably be objects outside both cones.



M_{sub} = {objects modifiable by sub}

O_{sub} = {objects observable by sub}

FIGURE 1

Condition (T) is a slightly stronger version of the Tranquility Principle of BL. In the absence of (T), certain covert storage channels described by Rushby [10] are permitted by the model. Essentially (T) implies the cones M_{sub} and O_{sub} do not change unless sub or an object in one of the cones is created or deleted.

The model also includes a set of rules for creating/deleting objects, granting/rescinding access rights of subjects to objects, etc. This set of rules constitutes the inputs for a finite state machine. The output and next state (state transition) functions are juxtaposed to form a results function. A system is said to be multi-level secure if any state which can be reached from a secure initial state is also secure. The theorem asserting a system is multi-level secure is known as the Basic Security Theorem [1]. It is proved by induction on the length of the string of operations performed. The mapping of an actual computer system to the elements of BL is fairly straight forward. Perhaps for this reason variations of it have been used in the design of several systems including MITRE's prototype [3], Ford's KSOS [9], SDC's KVM/370 [6], and Honeywell's SCOMP [3], the only one being developed as a commercial product.

III. The Feiertag-Levitt-Robinson Policy

The Bell-LaPadula policy has been criticized as too inflexible. Also, since it is an access control policy, it is not as amenable to automated verification techniques or to checks for covert channels as are flow control policies. The policy developed by Feiertag, Levitt, and Robinson, [4], denoted FLR, is a more flexible policy based on flow control. It was formulated to be used with SRI, International's Hierarchical Development Method (HDM) [8] for the design and verification of software. The components of the finite state machine are more immediate in FLR than in BL. These are:

1) The set of state variables. These correspond to the objects of BL. The state of the system consists of a value for each of the state variables.

2) The set of function references, that is, a function along with a set of parameter references for the function. These are the inputs to the finite state machine. They include the operations of observe and modify, but also include state changing rules analagous to those to those in the model of Part II.

3) The set of outputs to users which result from function references.

4) The next state and output functions which map a state and function reference pair to a state and output respectively. And

5) The set of security levels for state variables and function references. If each function reference is assumed to have a subject which invokes the function, then the level of the function reference can be equated with the level of the subject.

The formal definition of multi-level security involves quite a bit of notation, however an English language version of the policy is stated here.

Definition: A system is multi-level secure if the following statements are true:

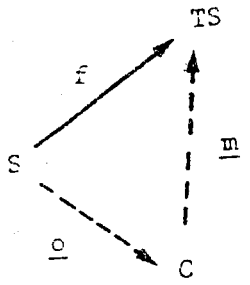
a) The results (output) of each function reference can be inferred from values of state variables at levels less than or equal to the level of the function reference.

b) After each function reference the values of all the state variables at a given level may be inferred from the values before the function reference of state variables at levels less than or equal to the given level.

c) Each function reference can only affect the values of state variables at levels at or above the level of the function reference.

It is fairly clear property (a) is the analog of (ss) from BL. Both (b) and (c) incorporate restrictions akin to (*). In fact (b) is the analog of an early version (*) [1].

The following example of the differences between BL and FLR is given in reference 4. Suppose a user (subject) at the confidential level, wished to modify the value of state variable (object) at the top secret level using the value of a state variable at the secret level. (See Figure 2.) This would violate (ss) of BL since the subject would have to observe the contents of the secret level object. On the other hand this would not violate any of the



BL.SS prohibits o

f is allowed by FLR

FIGURE 2

properties of FLR, if there were a function which would allow a subject to use one object to modify another object without observing the first object. In FLR the set of objects which may be used to modify another object is independent of the function performing the modification. This fact is illustrated in Figure 3. The cones of objects observable and modifiable by sub remain the same, but if obj is an object in M_{sub}, the cone, M_{obj}, of objects which can be used to modify obj may be larger than O_{sub}. This allows an upward flow of information from which no compromise is possible, since the subject initiating the flow is at a level less than or equal to the levels of any objects modified. From this discussion it is clear that in general the two policies, BL and FLR, are not equivalent. It is true that BL implies FLR. That is, any system which satisfies BL automatically satisfies FLR. This assertion is proved in the appendix.

Rushby states the following Object Monitor Axiom [10] which he later formalizes in the language of Goguen and Meseguer [11].

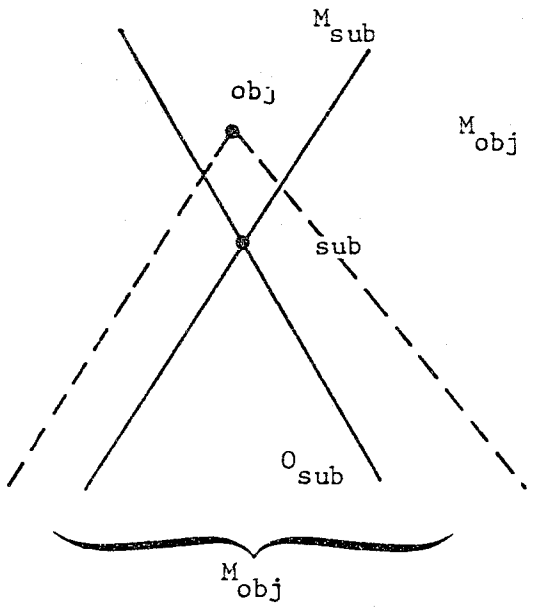
Definition: A system satisfies the Object Monitor Axiom (OMA) if:

a) The output of an operation may only depend on the values of objects to which the user who executes the operation has observation rights.

b) The new values assigned to objects as a result of executing an operation may only depend on the values of objects to which the user who executes the operation has observation rights.

c) An operation may only change the values of objects to which the user who executes the operation has modification rights.

He uses this axiom to show BL implies the Goguen and Meseguer version of MLS. In this paper a similiar set of assumptions are implicit in the proof that BL implies FLR. In fact, the similarity between OMA and FLR along with Figure 1, which is a consequence of (ss) and (*), constitutes the proof of the implication.



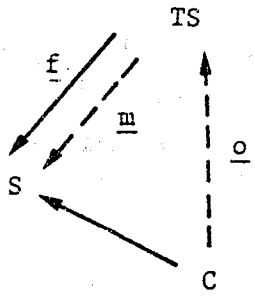
M_{obj} = {objects that can be used to modify obj}

FIGURE 3

This policy, FLR, defines multi-level security for the system rather than for a state of the system. This means an analog of the Basic Security Theorem is contained in the statement of the security policy. Hence, one would expect a proof that a model satisfies FLR to be less obvious than a similar proof that a model satisfies BL. On the other hand, it is not necessary to state and prove a separate Basic Security Theorem. The SCOMP, kernel was originally verified using a variation of FLR [3].

IV. The Goguen-Meseguer Policy

As motivation for the third policy, it is useful to contrast the situation illustrated by Figure 2 with the one illustrated by Figure 4, in which a user at the top secret level uses an object at the confidential level to modify an object at the secret level. This is prohibited by both the previous policies. In BL (*) prohibits the user from modifying a lower level object. In FLR (c) prohibits any function reference which would modify a state variable at a level which is not greater than or equal to level of the subject of the function reference. In both examples the overt flow of information is upward, which one might expect to be permissible. However in the second example the top secret user could signal information to a secret user with o access to the secret object by the manner in which the information in the confidential level object is used to modify the secret level object. Stated another way the difficulty with the second example is that a high-level user could potentially interfere with a low-level user. This presents the potential for a covert channel running from the high-level to the low-level.



BL.* prohibits m

FLR.c prohibits f

FIGURE 4

If the response of the system for each user were only dependent on the activities of users at or below the level of that user, then no information could flow from a high to a low-level either overtly or covertly. Moreover the mechanisms enforcing security would be essentially transparent to the user.

The policy of Goguen and Meseguer [5] is based on this notion of non-interference. Its notation and formalization are simpler than those of FLR, and its statement is more elegant than either of the previous policies. The elements of the model are:

1) The set of possible states of the system.

2) The set of users of the system.

3) The set of commands which the users may issue. The set of function references in FLR corresponds to the Cartesian product of the set of users with the set of commands.

4) The set of outputs to the users.

5) The next state and output functions.

6) The set of security levels for users.

Definition: User 1 is said to be non-interfering with user 2 if for any two strings of user-command pairs, which are identicial except for pairs involving user 1, the outputs to user 2 are identicial.

Definition: The system with a given initial state is multi-level secure, if user 1 is non-interfering with user 2, unless the level of user 1 is less than or equal to the level of user 2.

This policy is very similar to the general MLS policy formulated in reference 4. Any system which satisfies FLR automatically satisfies GM. The proof is contained in the appendix. Since no counterexample has been produced, it is possible GM is equivalent to FLR. It would be nice to have either a proof of their equivalence or a description of the differences between them. With GM as with FLR, an analog of the Basic Security Theorem is inherent in the definition of the policy. Since the definition of non-interfering involves arbitrary finite strings of user-command pairs, one approach to proving one user is non-interfering with another is to use induction on the length of the string. This means the mechanics of the Basic Security Theorem have not disappeared, they have simply been moved. Rushby [12] has developed a series of unwinding theorems which provide the inductive tools necessary to verify a model satisfies GM.

V. Conclusions

Three formal policies for multi-level security of a computer system have been examined and compared. Each policy is stated as a definition in terms of a finite state machine model of the system. The statements of the policies differ because the components of the machine are defined differently for each model and also because the control objectives differ for the three policies.

BL is based on access control. It is the most specific and appears to be the easiest policy to use in the design of a system. FLR controls information flows. It allows for greater flexibility in the manipulation of data and lends itself nicely to verification techniques utilizing automatic theorem provers. This is not surprising since it was developed for use with SRI's HDM. GM is based on the control of interference among users. It is the most general and most elegant of the three. It is relatively new, 1982, and has not been used for the design or verification of any system. It does provide a nice language for high-level discussions of formal security policies. It is possible to interpret the components of one model in terms of the components of another model. By doing this one can see the policies are not, in general, equivalent. Rather any system which satisfies BL also satisfies the other policies, and any system which satisfies any one of the policies also satisfies GM. All three policies appear to enforce the DOD mandatory security policy for the control of information accessible to users of the system.

Although the policies are generally distinct, there are conditions under which they are equivalent. These conditions reflect assumptions made on the behavior of the system or on the level of use being modeled. One such assumption is that there is no way for a process to use one piece of information to modify another piece of information without the user on whose behalf the process is acting acquiring the capability to observe something about the first piece of information. If the assumption is true, the three policies are equivalent. Proofs of this equivalence as well as of the other implications are contained in the appendix.

References:

1. Bell, D.E. and L.J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model", MITRE Corp, Bedford, MA (September, 1974).

2. Biba, K.J., "Integrity Considerations for Secure Computer Systems", MITRE Corp., Bedford, MA (April, 1977).

3. Bonneau, C.H., "Secure Communications Processor Kernal Software: Detailed Specification, Part I, Rev. D.", Honeywell, St. Petersburg, FL (1980).

4. Feiertag, R.J., K. N. Levitt, and L. Robinson, "Proving Multi-level Security of a System Design", in Proceedings, Sixth ACM Symposium on Operating Systems Principles, pp. 57-65 (1977).

5. Goguen, J.A. and J. Meseguer, "Security Policies and Security Models", in Proceedings, 1982 IEEE Symposium on Security and Privacy, pp. 11-20 (April, 1982).

6. Gold, B.D. et.al., "A Security Retrofit of VM/370", in Proceedings of the AFIPS National Computer Conference, Vol. 48, pp. 335-342, Arlington, VA, AFIPS Press (1979).

7. Grohn, M.J., "A Model of a Protected Data Management System", I.P. Sharp Associates Ltd., Ottawa, Canada (June, 1976).

8. Levitt, K., L. Robinson, and B. Silverberg, "The HDM Handbook", Computer Science Labortory SRI International, Menlo Park, Ca (1979)

9. McCauley, E.J. and P.J. Drongowski, "KSOS: The Design of a Secure Operating System", in Proceedings of the AFIPS National Computer Conference, Vol. 48, pp. 345-353, Arlington, VA, AFIPS Press (1979).

References: (continued)

10. Rushby, J.M., "The Bell and LaPadula Security Model", Draft report, Computer Science Laboratory, SRI International, Menlo Park, CA (1984).

11. Rushby, J.M., "Comparison Between the Bell and LaPadula and the SRI Security Models", Draft report, Computer Science Laboratory, SRI International, Menlo Park, CA (1984).

12. Rushby, J.M., "The SRI Security Model", Draft report, Computer Science Laboratory, SRI International, Menlo Park, CA (1984).

13. Schiller, W. L., "The Design and Specification of a Security Kernal for the PDP 11/45", MITRE Corp., Bedford, MA (March, 1975).

14. Silverman, J.M., "Proving on Operating System Kernal Secure", Honeywell, Minneapolis, MN (April, 1981).

15. Taylor, Tad, "Comparison Paper Between the Bell and LaPadula Model and the SRI Model", in Proceedings 1984 IEEE Symposium on Security and Privacy, PP 195-203, (May, 1984).

Formal Descriptions of the Policies and Proofs of the Implications

Throughout this appendix K will refer both to the set of security levels of the security level lattice and to the function associating a security level with each of the appropriate entities in the model of the system. Thus in BL function, K has as its domain the union of the sets of subjects and objects of the system. The range of function K is always the set K of security levels.

I. Description of BL

The primitive elements of the system are:

SUB = the set of subjects.

OBJ = the set of objects.

A = set of access modes. These modes are o and m.

These are used to define the state of the machine, which consists of

b = the subset of (sub, x, obj) in SUB x A x OBJ such that sub has x access to obj.

K = the security level function, and other components irrelevant to this discussion.

Definition 1. A state is secure if both of the following are true.

ss) Whenever (sub, o, obj) is in b, $K(sub) \geq K(obj)$.

*) Whenever (sub, m, obj) is in b, $K(sub) \leq K(obj)$.

The inputs to the machine are a set of requests to alter the state of the system including the creation and deletion of objects. The response of the system to each of these requests is a next state and a message to the subject, which made the request, based on the security level of the subject and the current state of the system. One state can be reached from another if there is a string of requests which transform the first state into the second via applications of the next state function with successive elements of the request string as inputs.

Definition 2. A system is secure if every state which can be reached from a secure initial state is also secure.

The Basic Security Theorem states that if each operation transforms a secure state into a secure state, then the system is secure.

II. Description of FLR

The elements of the model are

V = the set of state variables. These correspond to the objects of BL.

S = the set of states. An individual state, s in S, is composed of one value for each state variable, v in V. The value is denoted v(s).

I = the set of function references. A function reference is a function with a full set of parameter references. The set of function references, includes the observe and modify operations as well as the state changing operations of the model in I.

O = the set of outputs which result from the invocation of a function reference to a particular state. These correspond to what an individual user sees after invoking a function.

N_r and N_s, the output and next state functions. These map SxI into O and S respectively. For economy of notation later

$sf := N_s(s,f)$.

In order to state FLR several preliminary definitions are required.

Definition 1. If k is a security level in K, then for each state s in S,

a) s_k = set of state variables at level k.

b) $s_{L(k)}$ = set of state variables at levels less than or equal to k.

c) $s_{G(k)}$ = set of state variables at levels greater than or equal to k.

Definition 2. Given a security level k, define the following functions with domain S:

a) P_k which maps each s in S to s_k,

b) Q_k which maps each s in S to $s_{L(k)}$, and

c) E_k which maps each s in S to $(s_{G(k)})^C$.

P_k and Q_k may be thought of as projections of S onto sets S_k and $S_{L(k)}$ respectively, and E_k may be thought of as filtering $S_{G(k)}$ from S, where S_k, $S_{L(k)}$, and $S_{G(k)}$ are defined in the obvious manner.

The FLR policy can now be stated.

Definition 3. A system is secure if each of the following is true.

a) For each f in I and each k in K, there is a function, j, so that if s is in S, then

$$N_r(s,f) = j(Q_k(s)).$$

b) For each f in I and each k in K, there is a function, j, so that if s is in S, then

$$P_k(sf) = j(Q_k(s)).$$

c) For each f in I and each s in S,

$$E_{K(f)}(sf) = E_{K(f)}(s).$$

In (a) and (b) the expression A = j(B) should be interpreted as A can be inferred from B.

III. Description of GM

The elements of the model are:

S = the set of states.
U = the set of users.
C = the set of commands the users can issue. In terms of FLR, UxC = I.
O = the set of outputs to the users
do = the next state function, which has domain SxUxC and assumes values in S.
out = the output function, which has domain SxU and assumes values in O. Out (s,u) is the value the user u sees when the system is in state s.

Several definitions are useful in stating the policy.

Definition 1. $(UxC)^*$ is the set of finite strings $(u_1,c_1)\circ..\circ(u_n,C_n)$ where each (u_j,C_j) is in UxC.

Definition 2. Do* is defined on $Sx(UxC)^*$ inductively by

a) $do^*(s,u,c) := do(s,u,c)$, and

b) $do^*(s,w\ (u,c)) := (do^*(s,w),(u,c))$ if w is in $(UxC)^*$.

Definition 3. If s_o is the initial state of the system, w is in $(UxC)^*$, and u is in U, then

a) $[w] := do^*(s_o,w)$, and

b) $[w]_u := out(do^*(s_o,w),u)$.

Definition 4. If $w = (u_1,c_1)\circ..\circ(u_n,c_n)$ is in (UxC)* and u is in U, then $P_u(w) := w_1\circ..\circ w_n$

where for i = 1..n,

$$w_i = \begin{cases} (u_i,c_i) \text{ if } u_i = u. \\ \text{the null string if } u_i = u. \end{cases}$$

Definition 5. User, u', is interfering with user, u, if

$$[w]_u = [P_u(w)]_u$$

for some string w in (UxC)*.

With all this notation the security policy can be stated quite simply.

Definition 6. The SYSTEM WITH INITIAL STATE S IS SECURE if whenever u' is interfering with u, $K(u') \leq K(u)$.

Notice with this definition the system may be secure for some initial states but not for others. Also, as noted in the text, the mechanics of the proof of the Basic Security Thereom are contained in the proof that if K(u') K(u), then u' is not interfering with u.

IV. BL Implies FLR

Theorem. If a system is secure in the sense of BL, then it is secure in the sense of FLR.

Before proving the theorem, it is necessary to interpret the model for FLR in terms of the model for BL.

i) The state variables of FLR correspond to the objects of BL.

ii) Each function reference, f, has a subject, sub, and K(f) = K(sub).

iii) $N_r(s,f)$ corresponds to an o operation by the subject of f on a set of state variables, v_i for i = 1..n.

Proof:

a) If $N_r(s,f)$ depends on v_i for i = 1..n, then for i = 1..n

(sub, $\underline{c}$, v_i) is in b, where sub is the subject of f,

and so (ss) implies

$K(f) = K(sub) \geq K(v_i)$.

Therefore

$N_r(s,f) = j(C_{K(f)}(s))$.

c) If v is a state variable and f is a function reference for which v(sf) v(s), then (sub, $\underline{m}$, v) is in b, where sub is the subject of f.

And (*)implies

$K(f) = K(sub) \leq K(v)$.

Therefore if $K(f) \nleq K(v)$ then v(sf) = v(s).

That is $E_{k(f)}(sf) = E_{k(f)}(f)$.

b) If $k \ngeq K(f)$, then as a consequence of (c)

$P_k(sf) = P_k(s)$.

If k K(f), v is in V with K(v) = k, and v' in V is used by f to modify v, then,

$K(v') \leq K(f) \leq K(v)$ by (*).

Therefore regardless of the relation between k and K(f),

$$P_k(sf) = j(Q_k(s)).$$

It is interesting to notice (c) implies (b) if the following condition is satisfied.

H) The expression "use v_2 to modify v_1" means "observe v_2 and modify v_1".

This condition will be referred to later. Given this condition (H), (c) and (*) are equivalent. There is an earlier formulation of (*) which is equivalent to (b). The two (*) conditions are equivalent [1] if the security level of a subject is taken to be its current security level, defined as:

$$K(sub) = \max\{K(obj)/(sub,\underline{o},obj) \text{ is in } b\}$$

as opposed to its maximum security level. In this report subjects are assigned their current security levels. Thus if (H) is satisfied, (*), (b), and (c) are equivalent, and in figure 3, $M_{obj} = C_{sub}$ whenever K(obj) K(sub).

V. FLR Implies GM

Theorem. If a system satisfies FLR, it also satisfies GM for every initial state.

The correspondence between models is fairly direct.

i) Out(s,u) corresponds to $N_r(s,f)$ for f = (u,c) for some c in C, and

ii) Do(s,u,c) corresponds to $N_s(s,f)$ for f = (u,c) in UxC.

Proof:

Suppose u' is interfering with u. That is, for some w in (UxC)* and some function invocation f = (u,c) with c in C,

$$N_r(sw,f) \neq N_r(sP_{u'}(w),f).$$

If v is a state variable which affects the output, then

$$v(sw) \neq v(sP_{u'}(w))$$

and by (a)

$$K(v) \leq K(f) = K(u).$$

So w contains at least one function reference, g, with subject u', which changes the value of v, hence by (c)

$$K(u') = K(g) \leq K(v).$$

Thus

$$K(u') \leq K(u).$$

VI. GM Implies BL if (H)

Theorem. If (H) is satisfied the three policies are equivalent.

The interpretation here is fairly straight forward

1) If the value of obj affects the value of out(s,u), then (u, o, obj) is in b, and

2) If the value of obj is changed by do(s,u,c), then (u, $\underline{m}$, obj) is in b.

It must also be assumed that if u creates obj, then

3) K(u) = K(obj) and

4) (u,$\underline{o}$, obj) is in b.

Proof:

ss) Suppose (sub_1, $\underline{o}$, obj) is in b, and obj was created by sub_2. Then sub_2 is interfering with sub_1, so

$K(sub_1) \geq K(sub_2) = K(obj)$.

*) Suppose (sub_1, $\underline{m}$, obj) is in b and obj was created by sub_2. Then (sub_2, $\underline{o}$, obj) is in b, so sub_1 s interfering sub_2 Therefore

$K(sub_1) \leq K(sub_2) = K(obj)$.

Thus if (H) is satisfied, BL implies FLR implies GM implies BL. So the three policies are equivalent.

EXTENDING THE BELL & LAPADULA SECURITY MODEL

Dr. Ronald A. Gove

DoD Computer Security Center

In this paper we propose an extension of the Bell & LaPadula computer security model (henceforth referred to as BLP) as presented in [B&LP1,2,3,4]. By enlarging the state space of BLP we will show that the information flow concepts (non-interference statements) of Feiertag, Goguen and Meseguer, and Rushby ([Feier], [GM], and [Rush1]) can be included in the BLP model. Rushby's formal version of the Feiertag model is referred to as the SRI model and will form our point of reference. We will assume throughout this paper that the BLP model is secure. That is, the initial state is secure and the rules are in effect. In order to simplify the presentation we will assume a fixed set of objects at fixed security levels; i.e. the rule change_object_security_level will not be allowed, and we will not, in this paper, concern ourselves with the addition or deletion of objects. We also will not utilized the hierarchy concept of [B&LP4].

Recall the following terms from BLP:

S = set of subjects

O = set of objects

V = set of states

L = partially ordered set of security levels (order relation <)

A = $\{ \underline{r}, \underline{w}, \underline{a}, \underline{e}, \underline{c} \}$ = set of attributes

F = $\{ f_S, f_O, f_C \}$ = set of clearance functions

Recall that the elements of set A correspond to $\underline{r}$=read only, $\underline{w}$=read/write, $\underline{a}$=append (write only), $\underline{e}$=execute and $\underline{c}$=control. The clearance functions are the subject level f_S:**S**-->L, the object levels f_O:**O**-->L and the current subject level f_C:**S**-->L. The BLP states are of the form v=(b,f,M) where b is the current access set, f F, and M is the access matrix.

In order to formulate non-interference concepts it is necessary to enlarge the BLP model by introducing the notion of the contents of an object. Let "values" be an abstract set representing the contents of all objects in **O**. The contents of an object will be identified by means of functions mapping into "values", e.g. h:**O**-->values. In order to emphasize the dependence of b, f, and M on the state v we will use the following notation: If v=(b,f,M) then,

access[v] = b

current_subject_level[v] = f

matrix[v] = M

Note that, since our model will not allow the maximum level of subjects and objects to change, we have assumed that f is just the current subject level function: f_c. Finally, we define a new state space for the extended BLP model, BLP*:

$$\mathbf{V}^* = \{(v_1, v_2, v_3, v_4) : (v_1, v_2, v_3) \in \mathbf{V} \text{ and } v_4 : O \rightarrow \text{values}\}.$$

Note: The contents function v_4 is state dependent; different states can have different contents. Following the notation used above, v_4 will usually be written as contents[v].

The BLP model affects state transitions by means of request and rules of operation. We will reinterpret these concepts through the concept of "commands" in order to follow the Rushby formulation of the SRI model. The command set, **C**, will contain two types of commands: the requests of BLP which act only on the access portion of the states and the operations which act only on contents[v]. The way the commands act on the states is through the state transition function next: $\mathbf{V}^* \times S \times \mathbf{C} \rightarrow \mathbf{V}^*$. For example, a get_read request of BLP would be represented by a command c=(get_read,o). In this case, next(v,s,c) would represent the next state, when subject s requests read access to object o when the system is in state v. For reference we will list the applicable requests of BLP and show the effect when those requests are granted. When requests are not granted, the next state is always unchanged. We will denote the current state by v and the next state by v*. State components that are not mentioned are not changed.

Requests 1, 2, 3, 4: get_read, get_append, get_execute, get_write

Semantics: subject s requests x-access to object o, where x is r, a, e, or w.

Effect: $\text{access}[v^*] = \text{access}[v] \cup \{(s, o, x)\}$.

Request 5: release_read/write/execute/append

Semantics: subject s requests x-access to object o be deleted, where x is r, a, e, w.

Effect: $\text{access}[v^*] = \text{access}[v] \hookleftarrow \{(s,o,x)\}$.

Request 6: give_read/write/execute/append

Semantics: subject s gives subject t x-access to object o, where x = r, w, e, or a.

Effect: for $(u,p) \in \mathbf{V}^* \times O$,

$$\text{matrix}[v^*](u,p) = \begin{cases} \text{matrix}[v](u,p) & \text{if } (u,p) \neq (t,o) \\ \text{matrix}[v](t,o) \cup \{x\} & \text{if } (u,p) = (t,o). \end{cases}$$

Request 7: rescind_read/write/execute/append

Semantics: subject s removes $\underline{x}$-access to object o from subject t, where $\underline{x}$ = $\underline{r}$, $\underline{w}$, $\underline{e}$, or $\underline{a}$

Effect: for $(u,p) \in V^{*x}\ O$,

$$\text{matrix}[v^*](u,p) = \begin{cases} \text{matrix}[v](u,p) \text{ if } (u,p) \neq (t,o) \\ \text{matrix}[v](t,o) \hookleftarrow \{x\} \text{ if } (u,p) = (t,o). \end{cases}$$

Request 8 and 9 of BLP deal with the creation and deletion of objects. Since we have omitted this aspect of the BLP model in this paper, we have no need to list them here. Request 11 is also omitted because it deals with changing object levels which we have also omitted. Request 10, "change_subject_security_level", requires some modification in order to fit our enlarged state space.

Request 10: change_current_subject_level

Semantics: subject t requests that its current clearance level be changed to L.

If the request is not in the correct form the system responds with ? and leaves the state unchanged. Otherwise, the following conditions are checked.

(i) $f_S(t) > L$

(ii) the star property will hold in the new state. i.e.

$(t,o,a) \in \text{access}[v] \Longrightarrow f_O(o) > L$

$(t,o.w) \in \text{access}[v] \Longrightarrow f_O(o) = L$

$(t,o,r) \in \text{access}[v] \Longrightarrow L > f_O(o)$

(iii) for any $s \in S$ such that $\sim(\text{current\_subject\_level}[v](t) < f_S(s))$, it is the case that $\sim(L < f_S(s))$

If any of these conditions fail, the rule give a "no" decision and the state v is unchanged. If they are all true then the new state is v^* where for any $s \in S$,

$$\text{current\_subject\_level}[v^*](s) = \begin{cases} \text{current\_subject\_level}[v](s) & s \neq t \\ L \text{ if } s = t \end{cases}$$

$$\text{access}[v^*] = \text{access}[v]$$

$$\text{matrix}[v^*] = \text{matrix}[v].$$

It is clear that next preserves the simple security properties and the * - property. Thus **V*** is secure if **V** is.

The "operation" or contents altering commands will be left undefined. Instead we will give an axiom that characterizes their behavior.

AXIOM I

Let $v \in \mathbf{V}^*$, $o \in \mathbf{O}$, $s \in \mathbf{S}$, $c \in \mathbf{C}$, and $v^* = next(v,s,c)$.

If access[v] does not contain (s,o,$\underline{w}$) or (s,o,$\underline{a}$) then
contents [v*](o) = contents[v](o).

NOTE: In words this says that if in state v user s does not have write or append access to object o then no command by s can change the contents of o.

In the SRI model an "output" function is needed in order to talk about information flows. Its exact formulation really does not matter as long as it satisfies the following condition: if a user, s, has read or write/read access to exactly the same set of objects when the system is in state v or in state u then the outputs must be identical. We need some additional notation to describe this precisely. Let $v \in \mathbf{V}^*$. Given $s \in \mathbf{S}$ we form the local (or subject) contents function lc[v,s]: **O** --> values by the rule

$$lc[v,s](o) = \begin{cases} contents[v](o) \text{ if } (s,o,\underline{r}) \text{ or } (s,o,\underline{w}) \in access\ [v] \\ \# \text{ otherwise} \end{cases}$$

NOTE: In words, the local contents of o with respect to subject s and state v is the contents of o if s has read access or write/read access to o and is the null, an undefined contents, #, if s does not have read or write/read access.

We then characterize the output function as any function out: **VxSxC** --> outputs (outputs an arbitrary set) that staisfies the following axiom:

AXIOM II

Let v, $u \in \mathbf{V}$, $c \in \mathbf{C}$, $s \in \mathbf{S}$, $v^* = next(v,s,c)$, and $u^* = next(u,s,c)$.

If lc[v,s] = lc[u,s] then

a. out(v,s,c) = out(u,s,c)

b. contents[v*] = contents[u*].

NOTE: In words, if in states v and u, s has $\underline{r}$ or $\underline{w}$ access to exactly the same set of objects with the same contents then the outputs have to agree, and the contents function of the next states have to agree.

Now we can define a security policy for BLP* as in the SRI model. Following [Rush1], we state the policy in terms of non-interference assertions denoted t-/->s (read:subject t does not interfere with subject s). Informally, t-/->s if nothing that t does can affect the "view" s has of the system. (This is developed formally in [Rush1], we omit the details). The security policy for the system is set, $P \subset S^x S$ with certain properties. The BLP* system will be secure in the SRI sense if for each $(t,s) \in P$ it can be shown that t-/->s.

The security policy, P, we want is:

$(t,s) \in P$ if

1. $t \neq s$

2. $\sim (f_S(t) < f_S(s))$

and 3. If v_0 is the initial state then $\sim$(current_subject_level[v_0](t) < f_S (s))

In words, (t,s) P if the maximum level of t dominates (or is non-comparable to) the maximum level of s and in the initial state, v_0, the current subject level of t dominates (or is non-comparable to) the maximum level of s.

In order to show that BLP* is secure in the SRI sense, we have to show that $(t,s) \in P$ implies that t-/->s. This will be done via Theorem 6 of [Rush1], the "unwinding theorem". We first need to show that condition 2 above also holds for all reachable states as well as the initial state v_0. (A state w is reachable from v if there is a sequence of commands c_1, $c_2 \ldots c_k$ and subjects s_1, $s_2 \ldots s_k$ such that w = next (...next(next(v, s_1, c_1), s_2 c_2)..., s_k, c_k)).

Lemma 1. If $v \in V^*$ and t, $s \in S$ and if $\sim$(current_subject_level [v](t) < f_S (s)), then for any $r \in S$ and $c \in C$,

$$\sim (\text{current\_subject\_level [next (v,r,c)]}(t) < f_S(s)).$$

Proof. An examination of the rules shows that the only command that affects the current_subject_level function of a state is command c: change current subject level to L. If next leaves the state v fixed we are done trivially. If next changes the state, then from rule 10 above

$$\text{current\_subject\_level [next(v,r,c)]}(t) = \begin{cases} \text{current\_subject\_level [v]}(t) \text{ if } t \neq r \\ L \text{ if } t = r \end{cases}$$

In the case $t \neq r$ we are done as current_subject_level[next(v,r,c)] = current_subject_level[v]. If t = r, current_subject_level [next(v,r,c)](t) = L and by condition iii of rule 10 (which must hold since next was assumed to have caused a state change) we conclude: $\sim (L < f_S(s))$.

By induction we have,

Lemma 2. If $v \in V^*$, t, $s \in S$ and $w \in V^*$ is reachable from v and $\sim$(current_subject_level[v](t) < f_S(s)) then

$\sim$(current_subject_level[w](t) < f_S(s)).

In order to apply theorem 6 of [Rush1] we need to specify an interpretation of the abstract view of the system. Let $v \in V^*$ and $s \in S$. Then the "view" s has of the system in state v is simply lc[v,s]; just the local contents function for s in state v. We will use view[v,s] and lc[v,s] interchangebly. It follows trivially from Axiom II above that the view function is internally consistent. That is, if view[v,s] = view[u,s] then out(v,s,c) = out(u,s,c).

We now state Rushby's

Theorem 6: Let M_R=(**V***, **S**, **C**, P, next, out, view) be an internally consistent system such that P is not empty and for all v, u ε **V***, s, t, r ε **S**, c ε **C**

1. (t,s) ε P ==> view[next(v,t,c),s] = view[v,s]

2. view[v,t] = view[u,t] => view[next(v,r,c),t] = view[next(u,r,c),t]

then M_R is secure (in the SRI sense).

We will show the Rushby-type system that we have constructed out of BLP satisfies the conditions of Theorem 6. This is done in lemmas 3 and 4.

Lemma 3. If v_0, v ε **V***, s, t ε **S**, (t,s) ε P, c ε **C** and v is reachable from the initial state v_0, then

view[next(v,t,c),s] = view[v,s]

Case 1 c is an operation.

In this case access[v] = access[next(v,t,c)] since operations can only change the contents function.

Let o ε **O** be an arbitrary object

subcase 1.1 (s,o,r) and (s,o,w) ∉ access[v]

It follows immediately that (s,o,r) and (s,o,w) ∉ access[next(v,t,c)] and view[v,s](c) = l[v,s](o) = # = lc[next(v,t,c),s] = view[next(v,t,c),s](o).

subcase 1.2 (s,o,r) or (s,o,w) ε access[v]

Without loss of generality, assume (s,o,r) ε access[v]. Then also, (s,o,r) ε access[next(v,t,c)] and lc[v,s](o) = contents[v](o) and lc[next(v,t,c),s](o) = contents[next(v,t,c),s](o)

claim: contents[next(v,t,c),s](o) = contents[v](o)

This will follow from Axiom I if we show that access[v] does not contain (t,o,w) or (t,o,a). Otherwise assume (WLOG) (t,o,w) ε access[v]. As we have assumed that BLP started in a secure state, v is a secure state and so the * - property holds for v. Thus (t,o,w) ε access[v] implies that $f_O(o)$ = current_subject_level[v](t). Since the simple security condition also holds and (s,o,r) ε access[v], $f_S(s)$ > $f_O(o)$. We conclude that $f_S(s)$ > current_subject_level[v](t). But this contradicts lemma 2 and the claim is proved.

From the claim and the definition of view subcase 1.2 is proved.

Case 2 c is a request.

subcase 2.1 c is a request of type 1, 2 or 3. Then access[next(v,t,c)] is just access[v] with the addition of at most (t,o,x), x A. Thus (s,o,r) or (s,o,w) are in access[v] if and only if they are in access[next(v,t,c)] and the

local contents is defined by contents in both states. Since requests cannot change contents the view of o is the same in both states.

subcase 2.2 c is a request 5. Then access[next(v,t,c)] is access[v] with, at most, the removal of (t,o,x). As t ≠ s the proof goes through as in subcase 2.1.

subcase 2.3 c is request 6 or request 10

access[v] = access[next(v,t,c)] and the proof is like subcase 2.1

subcase 2.4 c is request 7

similar to 2.2

This concludes the proof of the lemma.

Lemma 4. view[v,s] = view[u,s] ==> view[next(v,t,c),s] = view[next(u,t,c),s]

Proof: The hypothesis is that lc[v,s] = lc[u,s]. From axiom II we conclude that contents[next(v,t,c)] = contents[next(u,t,c)]. The views will be the same if we show that s has the same access to all objects in both states next(v,t,c) and next(u,t,c).

Case 1 c is an operation. Then access[v] = access[next(v,t,c)] and access[u] = access[next(u,t,c)]. Since s has the same access in states u and v, by hypothesis s has the same access in states next(v,t,c) and next(u,t,c).

Case 2 c is a request. Since the access for s are the same in both states v and u, each request will be granted or denied the same way in each state and the next state will have the access changed identically.

Theorem 1. The Rushby MLS policy, P, defined above is secure.

Proof: Lemmas 2 and 3 give the hypothesis of Rushby's theorem 6 and the conclusion follows.

[B&LP1] Bell, D. Elliott and LaPadula, Leonard J., "Secure computer Systems: Mathematical Foundations", MTR-2547 (ESD-TR-73-278) Vol I, The MITRE Corporation, Bedford Massachusetts, 1 March 1973.

[B&LP2] Bell, D. Elliott and LaPadula, Leonard J., "Secure Computer Systems: A Mathematical Model", MTR-2547 (ESD-TR-73-278) Vol II, The MITRE Corporation, Bedford Massachusetts, 31 May 1973.

[B&LP3] Bell, D. Elliott, "secure Computer Systems: A Refinement of the Model", MTR-2547 (ESD-TR-73-278) Vol III, the MITRE Corporation, Bedford Massachusetts, December 1973.

[B&LP4] 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.

[Feier] Feiertag, R.J., "A Technique for Proving Multilevel Systems Secure", Tech Report CSL109, SRI International, January 1980.

[GM] Goguen, J.A. and Meseguer, "Security Policies and Security Models", Proc. 1982 Syposium on Security and Privacy, Oakland California, IEEE Computer Society, April 1982.

[Rush1] Rushby, John, "Mathematical Foundations of the MLS Tool for Revised SPECIAL", to be published.

[Rush2] Rushby, John, "The Bell and LaPadula Security Model", to be published.

The Security Model of Enhanced HDM

John Rushby

Computer Science Laboratory
SRI International

Abstract

The Enhanced HDM Specification and Verification System being developed at SRI International includes an "MLS Checker" that automatically verifies the security of a certain class of system specifications.

This paper gives a brief and informal overview of the security model on which the MLS checker is based and discusses its application and its relationship to other security models and to the requirements of the DoD Trusted Computer System Evaluation Criteria.

1. Introduction

SRI's Enhanced HDM Specification and Verification System will include a subsystem known as "The MLS Checker" that determines whether system specifications are consistent with the DoD Multilevel Security (MLS) policy. In order to do this, the MLS Checker embodies certain assumptions about:

- The "meaning" of specifications written in Revised Special (this is the specification language of Enhanced HDM),
- The sort of systems whose specifications are to be checked, and
- The interpretation of "security" that is appropriate to that class of systems.

The first of these assumptions concerns the semantics of Revised Special and will not be discussed here; the other two assumptions constitute the ***security model*** of Enhanced HDM and are the subject of this paper. The security model of Enhanced HDM is the same as that of "Old" HDM, which was developed by Feiertag, Levitt, and Robinson in 1977 [3] and which provided the basis for the original MLS Checking Tool developed by Feiertag [4]. The description of the model has been improved over the years (notably by Goguen and

Meseguer [5]); the informal presentation given here is based on the current technical description [11]. It should be stressed that it is only the MLS Checker that has this (or any other) security model built into it; the rest of the system is a general specification and verification environment that may be used to state and verify arbitrary system properties – including those derived from other security models.

Security models are abstract descriptions of computer systems that concentrate on matters relating to the protection and security of information. Such models are helpful in two aspects of the system design process: *synthesis* and *analysis*. By emphasizing just the features relevant to security, a security model can serve to clarify and guide the design (i.e. synthesis) of a secure system; and, by providing a formal basis for the notion of "security", a model can provide the foundation needed to conduct a rigorous informal analysis, or a formal verification, of the security of a system design. Because of its application to the design of the MLS Checker, it is this second aspect that is emphasized in the security model for Enhanced HDM.

As indicated above, a security model has two components: the first embodies assumptions about the sort of systems that are to be considered, while the second defines a notion of security that is appropriate to that class of systems. I will call these, respectively, the ***system*** and the ***security*** components of the model. The ***utility*** of a model is closely related to the realism of the assumptions that constitute its system component; the ***correctness*** of a model is a function of its security component.

In order to understand what is meant by the "utility" and "correctness" of a model, it is necessary to understand how a system is verified with respect to a security model. Essentially, verification consists of a demonstration that the system is a valid *interpretation* of the model; that is to say, one must establish a correspondence between the elements of the system and those of the model and must show that the elements of the system interact only in ways that are consistent with the model. A model is of limited utility if few, if any, systems of interest can be shown to be consistent with it; a model is incorrect if a system that has been shown to be consistent with the model fails, nonetheless, to meet its security requirements.

Since one of the main purposes of a security model is to capture security requirements formally and unambiguously, there will generally be no independent formal description of a

system's security requirements against which to evaluate the correctness of its model. Furthermore, testing is a notoriously unreliable technique for discovering subtle flaws in a system, and this is especially true in the case of security – where flaws may become manifest only under conditions of sophisticated and deliberate abuse. Thus there is no reliable way of determining whether a system meets its security requirements other than by verifying its compliance with a formal security model. It follows that incorrect security models cannot be countenanced: they just *have* to be right.

The best way to ensure that a model is correct is to make it so simple that it can be *totally* comprehended by suitably skilled persons. In this way, the correctness of a model can be established by the social process of peer review – just as the correctness of a mathematical theorem is established. This requirement for simplicity argues for very abstract security models: ones from which all irrelevant issues, and all details peculiar to a given system, have been stripped away so that the single issue of security is isolated and exposed to scrutiny.

Unfortunately, the desire for highly abstract security models conflicts with one of the realities concerning their application. In practice, the demonstration of consistency between a system and its security model is rarely accomplished in a single step – the "gap" between them is just too great to bridged so simply. (The details of the interpretation would be so complex that they would, themselves, be prone to error). Instead, it is generally an abstract ***specification*** of the system that is verified with respect to a security model. The step of showing that the actual system is a valid interpretation of its verified specification is generally performed informally. (Formal techniques do exist, but they are hugely expensive). The informality, and possible unreliability, of this second step raises the possibility that undetected security flaws may be introduced during the implementation of a secure specification. In order to reduce the likelihood of such flaws, it is desirable that the specification should be "close" to the implementation – so that complexity of the informal verification step is reduced as much as possible. In particular, the security mechanisms to be employed in the implementation should be described, in all essential details, in the specification. Formal verification of the specification with respect to a security model, followed by informal verification of the system with respect to the verified specification, then gives considerable confidence in the security of the final system.

The desire to verify detailed, concrete, system specifications argues for a detailed, highly concrete, security model – since otherwise the interpretation from model to specification becomes complex and error-prone and we will be back where we started. Unfortunately, however, such concrete security models often raise the very doubts they are meant to allay: if the model is highly detailed and its definition of security correspondingly so, then one naturally wonders whether that definition is correct. The discovery of subtle quirks and outright flaws in certain well established security models shows that these doubts are not idle [9, 15].

Thus we are confronted with a dilemma: in order to be sure that it is itself correct, a security model should be simple and highly abstract; but in order that it can be used to verify usefully detailed system specifications, a model must be detailed and concrete. In my view, the correct escape from this dilemma is through the horns: rather than argue that abstract models are superior to concrete ones, or vice versa, we should recognize the need for *both*. Having said that, however, I also claim that abstract models should be given primacy. My reason is that ***abstract models can be used to verify the correctness of concrete ones*** – this is accomplished by showing that the axioms of the abstract model are provable as theorems of the more concrete one. Furthermore, although abstract models cannot be used directly to verify the security of a detailed specification of a system's *implementation*, they have a useful role to play in verifying an abstract specification of its *interface*. This is useful in its own right, since a system may be unsecure for (at least) two reasons: either its specification may include inherently unsecure operations (i.e., its ***interface specification*** may be unsecure), or its mechanisms may fail to correctly implement otherwise secure operations. It is somewhat heavy-handed to detect flaws of the first kind using models intended to detect those of the second kind. For example, it is surely better to discover immediately if a file system contains operations that permit unclassified users to read classified files, rather than wait until the internal mechanisms of the file system are found to be inconsistent with a concrete security model. An abstract security model can be used to perform the useful task of verifying the security of system interface specifications before resources are committed to its implementation.

The security model of Enhanced HDM is a highly abstract one. Its merits are its simplicity and elegance: it is easy to see that it is correct (as far as it goes – and I will discuss the issue of its completeness later). Its applications are the verification of system interface specifications

(performed automatically using the MLS Checker of Enhanced HDM), and the verification of more concrete security models (currently performed by hand). An informal description of the HDM model is given in the next section; technical details may be found in [11].

2. The Model

As explained in the introduction, there are two components to a security model: the system and the security components. The system component of the HDM security model is a conventional finite automaton. That is to say, a computer system is regarded as a "black box" that consumes input "tokens" one at a time and, with each token consumed, changes its own internal state in a manner that depends upon its current state and the value of the input token consumed. At the same time, an output token, whose value is determined in the same way, is emitted and returned to the user who sent the input token that initiated the activity. The internal state of the system is not visible outside; all that can be observed is its input/output behavior.

This automaton model captures the essential characteristics of many types of systems, and system components, quite accurately. Consider, for example, a file server that receives requests from users to save and retrieve files and that returns files and status information to users in response to those requests. The requests sent to the file server can be identified with the input tokens of the model, while the results that it returns can be identified with the output tokens. The internal state of the file server consists of the file system that it maintains. The receipt of a request from a user will cause it to update the file system on the basis of information sent with the request, and to return a result determined by the contents of the file system and the nature of the request.

We now need to add a security component to this system model. The first step is simply to interpret the system model a little differently. Instead of tokens being sent by, and returned to, human users directly, we now recognize that those users may be supported by untrusted computer systems or processes. Whereas a human with legitimate access to classified information may be trusted not to reveal that information to unauthorized persons, a computer system cannot. We indicate that the users of the system are now identified with untrusted computer systems or processes by using the term ***subject*** instead of "user". We associate a ***sensitivity label*** with each subject to indicate the ***clearance*** of the (human) user identified

with that subject. The sensitivity labels are assumed to be partially ordered by a ***dominates*** relation[1] that defines the ***security policy*** to be enforced by the system. This policy requires that no information may flow from one subject to another unless the clearance of the recipient dominates that of the sender. The heart of the security component of the HDM security model is the way in which it gives a precise definition to what it means for information to "flow" from one subject to another.

This definition is beautifully simple: an input from one subject causes ***information to flow*** to another subject if the outputs subsequently seen by the second subject are *different* from those that he would have seen if the input concerned had not been present. The security component of the HDM model then simply requires that information may flow, in the sense just defined, from one subject to another only if the clearance of the recipient dominates that of the sender.

In the case of the file server example, a request to delete a file obviously causes information to flow to all those subjects who may subsequently determine that the file has been deleted. But our definition of information flow is much stronger than this: it says that information flows to all subjects for whom the file server will subsequently behave *differently* than it would have done if the delete-file request had not been issued. Thus, if the delete-file request causes some disk space to become free and another subject can subsequently discover that the amount of free space has changed, then information has flowed to that subject from the one that sent the delete-file request – and this is to be allowed only if the clearance of that subject dominates that of the subject that sent the delete-file request. It can be seen that this *information-flow* characterization of security is very powerful: it embraces covert storage channels (though not timing channels) as well as direct disclosure.

I claim that this formulation of what is meant by security captures clearly, and correctly, the intent behind other formulations of the property. There is, however, a serious charge that can be brought against the model. The charge is that a valid security model should be based on established government regulations regarding the handling of classified information. This

---

[1]Security Level S_1 is said to dominate security level S_2 if the hierarchical classification of S_1 is greater than or equal to that of S_2 and the non-hierarchical categories of S_1 include all those of S_2 as a subset [2, p110].

would suggest the introduction of *objects* as the repositories of information "within" the system state and, by analogy with the regulations of the "pen and paper" world, we should demand that objects be labelled with the sensitivity level of their contents (i.e. their ***classification***) and that subjects may only read objects whose classifications are dominated by their own clearances. My objection to this approach is that the regulations that would be taken as the starting point in the construction of our model are not a statement of the *intent* of security procedures, but are a particular set of *mechanisms* that are appropriate for safeguarding security in the "pen and paper" world. In effect, they are a security model whose (unstated) system component is a set of assumptions about the way in which the "pen and paper" world operates. Computer systems do not correspond to these assumptions (they are not passive entities like paper and vaults) and this invalidates the security component of the "pen and paper" model. That this is true is manifest by the need to introduce additional axioms (e.g. the "*-property" [1]) into those computer security models that follow this approach.

The attempt to base computer security models closely on established regulations is a laudable one; my objection is to taking this approach as a *starting point* – for then we have no "higher-level" notion of security to appeal to in cases where these mechanisms prove inadequate. The "trusted processes" of the Bell and La Padula model [1] are a case in point. These processes are not constrained by the *-property because they are trusted not to violate the *intent* of that model property. The problem is then to establish the security of these processes in the absence of a precise description of just what the "intent" of the model is: because the Bell and La Padula model describes a particular set of security mechanisms, it provides no guidance in cases where its mechanisms prove inadequate.

Instead of identifying security with a particular set of mechanisms, I argue that it is preferable to first enunciate a principle of security that attempts to get at the *intent* behind such mechanisms. Once this has been done, we can introduce particular security mechanisms into our model – mechanisms based on government regulations – and attempt to verify them with respect to our more abstract model. If this attempt succeeds, then we have the satisfaction of knowing that two fairly independent efforts to formalize the same set of concerns have converged on the same (and therefore presumably correct) point; if it fails, then investigation of the discrepancy between the models will sharpen our understanding of the

issues concerned and may lead to the discovery, and subsequent correction, of errors in one or both of them. I will describe an experiment of this kind later.

Although I claim that the HDM security model, as formulated so far, provides a useful *definition* of security, it is too abstract to give much practical guidance in the construction or analysis of secure systems. The treatment of a computer system as a "black box" with no internal structure is not helpful to those who must design or analyze the internal mechanisms of a computer system. Furthermore, the definition of information flow is quantified over all *sequences* of future state transitions: that is, information is considered to flow from one subject to another if the presence of an input from the first subject can cause *any change whatever* in the subsequent behavior of the system as perceived by the second. What we would really like is a characterization of security that applies to *single* state transitions, rather than to sequences of transitions.

Both these deficiencies in the most abstract formulation of the model can be remedied by adding more structure to its system component. Instead of treating the system state as a "black box" with no internal structure, we will now assume that the system state is a record of the ***values*** held in ***objects***. We will further assume that each object is assigned a *fixed* sensitivity label called its ***classification***. We also need a little more terminology: we will say that the individual steps performed by the system are called ***operations***. An operation consumes an input token, causes a state change, and produces an output token. The sensitivity level of an operation is taken to be that of the subject that sent the input token that invoked it.

Given these elaborations to the basic model, it is possible to prove the following result.

Theorem: A system is secure if each of its operations satisfy the following three conditions.

1. The output produced by an operation at sensitivity level l depends only on the values of objects whose classifications are dominated by l.

2. An operation at sensitivity level l changes only the values of objects whose classifications dominate l.

3. If an operation changes the value of an object at classification l, then the new value assigned to the object depends only on the prior values of objects whose classifications are dominated by l.

This result is the one on which the MLS Checker is built. I will discuss its interpretation and application in the next section.

3. Applications

The theorem quoted at the end of the previous section provides the basis for the MLS Checker of Enhanced HDM, and for a similar tool developed earlier by Feiertag [4]. These tools process system specifications that have been augmented with information concerning the sensitivity labels associated with the objects and operations defined in the specifications and then check the specifications of the operations to see that they comply with the three conditions stated in the theorem. This checking involves the generation and proof of putative theorems (called ***verification conditions***) concerning relationships among the sensitivity labels of the operations and objects defined in the specification. The Checker also ensures that the specifications are sufficiently ***complete*** that they define the behavior of the system under *all* conditions. Individually, the conditions that need to be checked are conceptually simple, but they are so numerous and detailed that it is unreliable and uneconomic to attempt the process by hand – it is the existence of automatic MLS Checkers that make this a viable method of security analysis. SRI's original MLS Tool [4] has been used in the analysis of several operational systems, including Honeywell's SCOMP [14]. This section considers what is accomplished by such analysis.

The first point to note is that, as stated earlier, it is only the *interface specification* of the system that can be verified in this way. Because the model requires all objects to be assigned *fixed* classifications, it does not address one of the major problems faced in the development of secure systems: the design and verification of mechanisms that permit system resources to be multiplexed securely among entities belonging to different security classifications. In effect, the HDM model assumes that each security classification operates in its own virtual system. It is possible to modify the security model so that the security classifications of objects are not fixed (the Mitre Corporation has built flow analyzers that do this) but this does not completely solve the problem. Instead, the correct solution to the problem is, in my view, to deny that there is

one! Checking interface specifications for security is an important process in its own right. Ensuring that those interface specifications are implemented correctly is an equally important, but different, problem that is best approached as a separate issue using techniques (based on more concrete security models) that are specialized to that problem.

The next point to note is that the system component of the HDM security model assumes that input tokens (i.e. requests for operations to be performed) are "tagged" with their correct sensitivity label. As I have explained earlier, this is a reasonable assumption for certain "application level" systems, or system components, such as file servers. It is not a valid assumption, however, for really basic system components such as an operating system nucleus. This is the component at the heart of a security kernel that establishes and maintains "process isolation through the provision of distinct address spaces" as required by the DoD Trusted Computer System Evaluation Criteria [2] for Evaluation Classes B1 and above. Requests arriving at the interface of the operating system nucleus are not associated with sensitivity labels provided by some outside agency: it is the task of the nucleus to *establish* this association. Also, operations do not arrive at the nucleus in an orderly, one at a time, fashion. Instead, it is the responsibility of the nucleus to respond to asynchronous interrupts and to establish the orderly transmission of operation requests to the processes that it supports. In short, the nucleus creates an environment for its client processes that is consistent with the HDM security model, while it itself operates in a far more complex and demanding environment. Once again, I do not see it as a weakness of the HDM model that it does not address these issues – that is the task of other, specialized security models [6, 10].

Just as it does not address the issue of the secure implementation of verified interface specifications, nor the security problems of an operating system nucleus, so the HDM security model does not confront security issues other than disclosure through information flow. In particular, the *discretionary* aspects of security policy are not addressed, nor those of *inference* and *aggregation*. Neither are the problems of specialized systems and components such as *databases* and *downgraders* considered. Specialized models are needed in all these cases.

In summary, the HDM security model focuses on just one aspect of the general security problem – but within that limited domain, it does a pretty good job. For the rest, let us have

lots of specialized security models that each focus on an individual problem with comparable clarity and precision, and let us learn how to combine these different models in ways that give comprehensive guidance and assurance to the developers of trusted computer systems.

4. Comparison with other Models

One of the most influential of security models is the one developed by Bell and La Padula [1]. Recently, some security flaws have been found in the model [9, 15] – some of its ***rules*** have been found to admit covert storage channels. In this section, I will show how the attempt to verify the Bell and La Padula model with respect to the HDM model provides a systematic technique for detecting such flaws.

In order to verify the Bell and La Padula model, we must demonstrate that it is a valid interpretation of the HDM model. The details of this demonstration are quite complex, since the two models use different mathematical formalisms. The following is a very informal, outline description of the process; those who desire the technical detail are referred to the appropriate reports [12, 13].

The Bell and La Padula security model is a concrete one, in that it describes explicit security mechanisms. Basically, the system state is partitioned into two components: the ***value state*** and the ***protection state***. The first of these is the (usual) record of the values stored in objects, while the second records the current and maximum security level of each subject, the classification of each object, and the type of access each subject is allowed to each object. In the simplest case, only two types of access need be distinguished: ***read*** and ***write***. If a subject has read access to an object, then it may *use* the value of that object when computing the output of an operation or the value to be stored in an object; it may only *change* the value of an object if it has write access to that object. These mechanisms are assumed to be provided by the system's "hardware".

The operations of the system are divided into two classes: the (regular) ***operations***, and the ***rules***. Operations access only the value state and are constrained by the "hardware" in the manner described above; operations correspond to the ordinary functions like "add", "load" and "store" etc. Rules, on the other hand, access only the protection state; they perform functions such as "give this subject read access to that object", and "change the classification of this object to that level".

The security component of the The Bell and La Padula model identifies security with the following two (slightly simplified) conditions:

simple security property: a subject may have read access only to objects whose classifications are dominated by its own clearance, and

****-property:*** an untrusted subject may have write access only to objects whose classifications dominate its own clearance.

It is quite easy to prove that these two conditions imply those of the theorem given earlier – the simple security property (henceforth abbreviated to *ss-property*) guarantees the first and third conditions in the statement of the theorem, while the *-property guarantees the second. Thus we deduce that the Bell and La Padula model is consistent with the HDM model *in the case of the regular operations*. The *rules* are a different matter, however.

Bell and La Padula gave a representative set of rules (based on those found in Multics) and argued that they were secure because they preserved the ss- and the *-properties. However, covert storage channels have subsequently been discovered in some of these rules. (A channel in the rule *change-subject-current-security-level* is described in [9], channels in the rule *change-object-security-level* are described in [12, 15]). These channels arise because the ss- and *-properties only consider the problem of information flow through the *value* component of the system state – the possibility of information flow through the *protection state* is not considered explicitly.[2] If one attempts to prove that the rules of the Bell and La Padula model are secure with respect to the HDM model, then the system state of the HDM model must be identified with the conjunction of *both* the value and the protection states from the Bell and La Padula model and the proof fails because certain of the rules permit unsecure information flows through the protection state. Although I have described this process as one performed by hand, it is possible (though I haven't tried it, nor thought through all the details) that it could be accomplished mechanically by constructing a specification of the Bell and La Padula model in Revised Special and then submitting it to the MLS Checker.

---

[2] In fact, many of the rules perform checks additional to those necessary to preserve the ss- and *-properties. The effect of these checks is to prevent covert storage channels that would otherwise have arisen, but the model does not explain why these checks are necessary, nor how to construct them systematically. It is the inadequacy of the checks in the two rules named above that admit the covert storage channels.

The lesson to be learned from this exercise is that the construction of concrete security models is a difficult and error-prone task and that informal review may not be an effective technique for uncovering subtle problems or oversights in such models. (The Bell and La Padula model is nearly ten years old, yet Millen and Cerniglia, who attribute the discovery of the covert storage channel in the rule *change-subject-current-security-level* to P.S. Tasker of the Mitre Corporation, observe that this channel was found only "recently".)

As I noted earlier, the rules present in the Bell and La Padula model were based on functions found in Multics. In order to model other systems, it may be necessary to introduce different rules that correspond more closely to those present in the systems of interest. For some application areas, completely specialized security models have been developed (see [7, 9] for examples), and this trend is likely to continue as novel applications are contemplated. In all these cases, whether they are new variations on established models, or completely new models, it is highly desirable that some objective, formal analysis of their correctness should be undertaken. In many cases, it seems that part of this analysis can be accomplished by verifying these new models with respect to the HDM model.

In comparison with other security models, the HDM model is much more abstract: it has no security mechanisms built in. However, these other security models can often be viewed as more detailed elaborations of the HDM model. Establishing this connection formally is a good way to evaluate some aspects of the correctness of these other models.

5. Relation to the DoD Trusted Computer System Evaluation Criteria

The DoD Trusted Computer System Evaluation Criteria [2] require the use of a formal security model for Evaluation Class B2 and above. For Evaluation Class A1 and beyond, formal methods are required in the analysis of covert channels (Paragraph 4.1.3.1.3) and a combination of informal and formal techniques must be used to demonstrate consistency between the Formal Top-Level Specification (FTLS) and the model (Paragraph 4.1.3.2.2). The Glossary to [2] provides some guidance on what constitutes an acceptable security model:

> "... to be adequately precise, such a model must represent the initial state of a system, the way in which the system progresses from one state to another, and a definition of a "secure" state of the system.
>
> "... the model must be supported by a formal proof that if the initial state of the

system satisfies the definition of a "secure" state and if all the assumptions required by the model hold, then all future states of the system will be secure."

A theorem satisfying the requirement of the second paragraph in this quotation is often called a *Basic Security Theorem* after a theorem of that name due to Bell and La Padula. A significant criticism of this requirement is that a Basic Security Theorem says essentially nothing about security – as McLean [8] demonstrated by proving just such a theorem for a model that clearly violates any reasonable notion of "security".[3] In fact, it should be clear that if Φ is *any* effectively decidable property of the system state, then an analog to the Basic Security Theorem can be constructed for that Φ. As McLean observed, a Basic Security Theorem is really a property of the finite-state system model employed (in that states can be indexed to support proof by induction), rather than of the particular definition given for security.

Bell and La Padula actually made very modest claims for their Basic Security Theorem (and made no subsequent use of it after they had proved it). They observed merely that it established [1, p21]

> "the relative simplicity of maintaining security: the minimum check that the proposed new state is "secure" is both necessary and sufficient for full maintenance of security".

In my view, the *intent* behind the requirements stated in the DoD Criteria is sound, but the particular requirement for a Basic Security Theorem is poorly chosen. If I may be permitted to interpret the intentions of the authors of that document, I would say that their real requirement was for a *concrete* security model. A concrete model is one, such as that of Bell and La Padula, that describes particular security *mechanisms*, as opposed to the HDM model, which describes only security *policy*. A security mechanism must obviously maintain some state information (recording who may access what, and in what way), and not all states will be equally "secure". Thus, it is natural (indeed, necessary) for a concrete model to prescribe a set of "secure states" and a set of rules which are proven (by a Basic Security Theorem) to be sufficient to guarantee that all state transitions are secure-state-preserving.

---

[3] Basically, McLean turned the *-property around, so that subjects may transfer information from higher to lower classification levels.

The identification of a set of secure states and the proof of a Basic Security Theorem do not, however, guarantee that a model enforces a useful form of security – they simply establish the internal consistency of a set of security *mechanisms*. A separate (preferably formal) justification is required in order to establish that those mechanisms enforce a more abstract statement of required security *policy*. As I have already observed, the HDM security model will serve well in this latter capacity.

Given that the verification of compliance between an actual system and its FTLS will be performed only informally, the requirement that a concrete security model be used for the verification of the FTLS is entirely reasonable – for we certainly wish to be sure that the security mechanisms of the system are included in the formal stage of its analysis. Nonetheless, and as noted earlier, the security verification of interface specifications provided by the MLS Checker of Enhanced HDM can also make an important contribution to overall security assurance, especially since it is the only formal technique able to detect covert storage channels. It would seem that the DoD Computer Security Center accepts this view since the verification of the Honeywell SCOMP kernel was largely accomplished with the aid of the MLS Checker of "Old" HDM. Also, the HDM security model continues to apply in those cases where the mechanisms of a concrete model prove inadequate, and trusted process are found to be required. Clarification of the Center's requirements and guidelines on all these topics would be welcome.

6. Summary

I have given an informal description of the security model employed by the MLS Checker of Enhanced HDM. This model is a highly abstract one that has no particular security mechanisms built in. The model gives a precise, formal definition of an information-flow interpretation of security that covers covert storage channels as well as direct disclosure. The model is so simple that there can be no doubt about its correctness. The applications of the model are the verification of system interface specifications and the analysis of more concrete security models.

References

1. D.E. Bell and L.J. La Padula, "Secure Computer System: Unified Exposition and Multics Interpretation," Technical Report ESD-TR-75-306, Mitre Corporation, Bedford, MA., March 1976.

2. Department of Defense, Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria,* 1983, CSC-STD-001-83.

3. R.J. Feiertag, K.N. Levitt and L. Robinson, "Proving Multilevel Security of a System Design," *Proc. 6th ACM Symposium on Operating System Principles,* pp. 57-65, November 1977.

4. R.J. Feiertag, "A Technique for Proving Specifications are Multilevel Secure," Technical Report CSL109, Computer Science Laboratory, SRI International, Menlo Park, CA., January 1980.

5. J.A. Goguen and J. Meseguer, "Security Policies and Security Models," *Proc. 1982 Symposium on Security and Privacy,* Oakland, CA., pp. 11-20, IEEE Computer Society, April 1982.

6. B.A. Hartman, "A Gypsy-Based Kernel," *Proc. 1984 Symposium on Security and Privacy,* Oakland, CA., pp. 219-225, IEEE Computer Society, April 1984.

7. C.E. Landwehr, "A Survey of Formal Models for Computer Security," *Computing Surveys,* Vol. 13, No. 3, pp. 247-278, September 1981.

8. J. McLean, "A Comment on the "Basic Security Theorem" of Bell and La Padula," Informal Note, Naval Research Laboratory, 1983.

9. J.K. Millen and C.M. Cerniglia, "Computer Security Models," Working Paper WP25068, Mitre Corporation, Bedford, MA., September 1983.

10. J.M. Rushby, "Proof of Separability - a Verification Technique for a Class of Security Kernels," *Proc. 5th International Symposium on Programming,* Turin, Italy, pp. 352-367, M. Dezani-Cianaglini and U. Montanari, eds., Springer-Verlag Lecture Notes in Computer Science, Vol. 137, April 1982.

11. J.M. Rushby, "The SRI Security Model," Draft Report, Computer Science Laboratory, SRI International, Menlo Park, CA., July 1984.

12. J.M. Rushby, "The Bell and La Padula Security Model," Draft Report, Computer Science Laboratory, SRI International, Menlo Park, CA., February 1984.

13. J.M. Rushby, "Comparison between the Bell and La Padula and the SRI Security Models," Draft Report, Computer Science Laboratory, SRI International, Menlo Park, CA., February 1984.

14. J.M. Silverman, "Reflections on the Verification of the Security of an Operating System," *Proc. 9th ACM Symposium on Operating System Principles,* pp. 143-154, October 1983.

15. T. Taylor, "Comparison Paper between the Bell and LaPadula Model and the SRI Model," *Proc. 1984 Symposium on Security and Privacy,* Oakland, CA., pp. 195-202, IEEE Computer Society, April 1984.

A1 POLICY MODELING

Jonathan K. Millen

The MITRE Corporation
Bedford, MA

INTRODUCTION

Many formal models of security policy and secure systems have been created over the last ten years. Some were aimed at expressing the DoD security policy, so that a formal specification of a planned system could be shown to support that policy. Others were aimed at investigating more fundamental issues such as information flow and the propagation of discretionary access rights.

Recently, the question of the proper function and design of a model has come under intense scrutiny, largely because of the publication of the Department of Defense Trusted Computer System Evaluation Criteria [CSC83]. A formal model of the applicable DoD security policy is required for systems to be rated in the higher protection classes.

It has been recognized that no one model will serve the needs of all applications. Modifications and extensions in policy are necessitated by differences in the type of system, e.g., a general purpose operating system as opposed to a network, and by differences in the operating environment or classification system used.

Nevertheless, it is felt to be beneficial to have a model that addresses the policy stated for A1 systems, the highest class included in the Criteria. It is anticipated that such a model could be used in two ways. It could be the model of the security policy supported by a proposed TCB (Trusted Computing Base), as required in the Criteria. It could also serve as a "kernel" around which more elaborate models can be built.

A model with those objectives is necessarily constrained in style and applicability. First, its subject matter and content are constrained to express the A1 security policy as stated in the Criteria. Hence, it deals with subjects, objects, security classifications and categories, and must include a particular restriction on the ability of subjects to read or write objects on the basis of their respective security levels. Like the Criteria document itself, the model will be limited in application to general purpose operating systems. While it is possible to build a secure message system, data management system, guard system, or network switch on top of a secure general purpose operating system, one would expect the security policy in each case to have various unique features. In some cases they could be added to the A1 model in the form of a superstructure or concrete interpretation, but in other cases it may be more practical to construct a different model altogether.

The Criteria document suggests that the Bell-LaPadula model [BLP75] would be acceptable as a formal model. It is doubtful, however, that it is the best choice for a model for a new TCB to be submitted for A1 certification. One reason is that it is unnecessarily restrictive - it includes specific "rules" for system functions, which may be incompatible with the desired TCB functions, and it includes a Multics-directory-like object hierarchy. The set of rules "...is in no sense

Work supported by the Department of Defense under contract no. F19628-84-C-0001.

unique, but has been specifically tailored for use with a Multics-based information system design" [BLP75, p. 19].

To the extent that rules determine functionality, they can suffer from the kinds of security problems that crop up in formal specifications. In fact, one of the rules in [BLP75] has a built-in covert storage channel for compromising information. This channel, a variation of a well-known one, will be discussed in detail as an example of one of the more subtle pitfalls that model designers should look for.

The Change-Subject-Current-Security-Level Channel

There is a rule in [BLP75] called "change-subject-current-security-level" by which a subject can change its current security level to any value that will satisfy the simple security property and *-property. In particular, it is possible for a subject to downgrade itself, as long as its read accesses are only to objects at or below the new, lower level.

The channel works as follows. When the subject is at the initial, higher level, it can read one bit of classified information at that level, and then decide to get read access to a fixed lower-level object or not, depending on the value of that bit. After releasing read access to the higher-level object, it can then downgrade itself to the lower level. At this point, the subject is not supposed to know any higher-level information; yet, it can determine the value of the higher-level bit by testing whether it does or does not have read access to the lower-level object. The result can then be written into some lower-level object.

The problem is more serious than its single-bit version suggests. First, it might be possible to repeat it rapidly, perhaps hundreds of times per second, with successive bits. One can also devise more complex versions involving several lower-level objects, to transmit larger words at a time.

This channel works despite the assumption that the subject is "memoryless", i.e., it has no implicit memory of its own. The information has not been stored in the subject, but rather in the state of the system.

Of course, one could prevent this channel by requiring that the subject forget its accesses when it is downgraded. This is quite correct, but it is unfortunately excluded by the rule, which states that the access set "b" is unchanged.

Technically, a system can still avoid the channel and obey the model if it never permits a change-subject-current-security-level call by itself, but only allows the call in a compound request in which the subject first releases its accesses. But this device defeats what is probably the best argument in favor of having specific rules, namely, that they might act as design guidance to avoid security problems.

AN A1 MODEL

In designing a new model to address A1 security policy, the initial objective was to create a "minimum model", one that covered the Criteria requirements and nothing more. Consequently, the suggested A1 model below, while it resembles the Bell-LaPadula model in having subjects, objects, accesses, and a form of the *-property, does not have rules for specific functions or an object hierarchy.

As other authors have discovered, however, there is an overwhelming temptation for a model designer to add new features in response to perceived deficiencies in other models. The model described below has two innovations. One expands the treatment of trusted subjects in a way intended to be more flexible and effective; the other incorporates discretionary security into the mandatory security level.

Trustedness of subjects is dissected into a collection of separate privileges which must be inherited by the subject from the objects to which it has execute access. Objects possessing such privileges are required to have high integrity, enforced by a component of the security level.

Discretionary security is handled by adding user-list components into the security level. Since security levels can be changed only by privileged software, the effect is to prevent individual access control from being subverted by Trojan horses.

INFORMAL STRUCTURAL DESCRIPTION

System State

The system state consists of a set of subjects, a set of objects, and some functions defining their current status. Each subject and each object has a security level and a (possibly empty) set of privileges. Associated with each subject is a set of objects to which it has read access, a set to which it has write access, and another set to which it has execute access.

A security level has the following components: classification, category set, integrity class, integrity category set, distribution list, and contribution list. The partial ordering "dominates" of security levels is based on the ordering of each of the components. The third through fifth components are ordered inversely, i.e., a greater level has a smaller value in those components.

Transitions

A transition is a state change in response to a request from some subject, called the requestor. There will be security conditions defining restrictions on secure transitions as well as on secure states.

Some transitions create subjects or objects. Mathematical entities are never really "created", of course; this just means that the set of subjects or objects associated with the next state is larger. Subjects or objects can also be deleted. Note that, since every "existent" subject and object has a security level and other attributes, any creations or deletions imply a change in those components of the state as well.

SECURITY CONDITIONS

Several of the security conditions given below are waived for subjects having an appropriate privilege; those conditions are starred (*). Subjects inherit their privileges from the objects they execute. Privileged subjects and objects must have a particular integrity category, called "Trusted", in order that their trustworthiness may be preserved.

A subject can grant a privilege p to an object only if it has a special privilege (Create-p) to do so. In an effort to control the propagation of privileges, we require that no privilege can create itself, either directly or indirectly. (To ensure this, define a function "Create" such that Create(p) = Create-p, satisfying the restriction that, for any set A of privileges, Create(A) cannot be a subset of A.)

Secure State Conditions

The Read and Write conditions below are derived from [CSC83, section 4.1.1.4, p. 45].

* Read: The level of a subject dominates the level of any object to which it currently has read or execute access.

* Write: The level of a subject is dominated by the level of any object to which it has write access.

Privilege: The privilege set of a subject is included in the privilege set of any object to which it has execute access.

Trust: If a subject or object has any privilege, The integrity category component of its security level includes the Trusted category.

Secure Transition Conditions

Transition conditions are waived only when the requestor (rather than any other subject mentioned) has the appropriate privilege.

* Tranquility: The security level of a subject or object can not change. Note that a change has different effects on different components of the security level. Separate privileges may be required for changes in different components of the security level.

* Creation: The security level of a new subject or object dominates that of the requestor.

* Access change: Only the accesses of the requestor can be changed.

Privilege change: A privilege p can be entered into the privilege set of an object only if the requestor has the privilege Create-p.

SECURITY LEVEL COMPONENTS AND ORDERING

A security level has five components, the first two having to do with information sensitivity, the next two with integrity, and the last two with individual access control. A security level dominates another if its sensitivity and contribution list components are greater (or equal) and its other components are less (or equal). The general principle is that a higher security level implies a greater restriction on access.

The partial ordering for each component is given below with the component description.

Classification

Usually one of the following: Unclassified, Confidential, Secret, and Top Secret. However, eight classifications are required for some National Security applications, according to guidance in [CSC83]. The classifications given above are linearly ordered, Unclassified being the least and Top Secret the greatest.

Category Set

Individual categories vary with the community, but a given system should support at least 29 categories to represent document compartment markings, according to guidance in [CSC83]. Furthermore, some additional categories may be needed to represent dissemination controls and other distribution limiters. Category sets are ordered by set inclusion, the empty set being the least and the set of all categories the greatest.

Integrity Class

Because the integrity class is an inversely ordered component, a subject can only read from a higher or equal-integrity object and write into a lower or equal-integrity object, so copying and computational operations cannot increase integrity. This form of integrity control, using the Read and Write conditions on a "dual" or inversely ordered integrity classification, comes from Biba's "strict integrity" model [Bib77]. Incorporating an integrity component into the security level was done first in the I.P. Sharp Protected Data Management System Model [Gro76].

There is some support for the idea that security classifications also carry a connotation of integrity. This idea can be implemented by having integrity classes Unclassified through Top Secret, with the understanding that the integrity class is not necessarily equal to the security classification. Typically one would expect that the security classification dominates the integrity class.

Integrity Category Set

Integrity category sets are ordered by set inclusion just like (sensitivity) category sets. Because it is an inversely ordered component, copying and transformation operations can only reduce the set of integrity categories.

In this model, there is a "Trusted" integrity category, which is intended to be used for objects containing software that will be executed by privileged subjects.

It might be asked why the individual privileges could not be implemented as integrity categories. The Read condition would then require that subject could not have a privilege unless the object to which it had execute access also had that privilege. The problem is that all the objects to which the subject had read access would have to have that privilege as well.

Distribution List

This is the set of names of users who are permitted to read an object. Distribution lists are ordered by inclusion, the empty set being the least and the set of all users being the greatest.

Because the distribution list is one of the inversely ordered components of the security level, it follows from the Read and Write conditions that an object can be copied or transformed only into another object with a smaller or equal distribution

list. This prevents information from receiving a wider distribution than originally intended.

It may be surprising that a subject has a distribution list. The intent here is that the subject's distribution list represents a mode of operation during a temporary session, and it places an upper limit on the distribution of information it is currently handling.

There is no notion within this model of a particular user on whose behalf a subject is operating. In order for the distribution list to have the desired effect of limiting the users who can receive information, there is an assumption we have to make about how the system being modeled is interfaced with the outside world. We assume that an output device being operated by a user is an object (or more than one object); and its distribution list should include that user. This assumption would be included in the security requirements for a trusted login process. The role of users is explained further below in a separate subsection.

In view of the fact that the distribution list is part of the security level, and the security level cannot be decreased by an unprivileged or untrusted subject, there may be some question whether this mechanism satisfies the intent of "discretionary" security.

The guidance for discretionary security in [CSC83], which is extracted in turn from DoD regulations, mentions two points: access control on an individual basis, and need-to-know. The distribution list mechanism clearly qualifies on the first point. As far as need-to-know is concerned, one thing is certain: authorization of need-to-know cannot be left to a Trojan horse. We know that Trojan horses are a concern, because they were the rationale for introducing the *-property, which reappears in this model in the form of the Read and Write conditions. Only a trusted, specifically privileged process can be expected to reflect the intent of an appropriate user when the distribution list is expanded.

The difference between a mandatory label like classification and a discretionary one like the distribution list is that a specific system administrator or operator must be consulted to change the former, while the custodian or owner of the object has sufficient authority to change the latter. This security policy should be embodied in the specifications for the privileged software for each of those tasks. It is not embodied in the model because of the difficulty of capturing the intent of a user. In the model, a user could be identified as an owner of each object, but this was not done because there is no formal axiom that explains the meaning of the relationship.

The term "discretionary" is confusing here, because changes in the security level are normally the province of "mandatory" or "non-discretionary" control. Perhaps the distribution list mechanism should be referred to as "individual" control instead.

Another possible objection is the apparent need to specify users one-by-one on the distribution list; The Al security policy calls for the ability to specify access for whole groups at a time. The apparent discrepancy is due merely to the level of abstraction of the model. A system implementing this model can specify sets of users symbolically in any desired way, as long as it is clear which individual users are included.

Contribution List

The contribution list is a set of users, like the distribution list, ordered by set inclusion. It is intended to implement individual control on write access. As in the case of the distribution list, we need an assumption about the external interface: the name of any user operating an input device must be on the contribution list of the object representing that device.

The Read and Write conditions imply that the contribution list of any object contains all users who may have influenced it, or will be permitted to influence it in the future.

INPUT, OUTPUT, AND USERS

Users were previously mentioned in the context of individual access control as elements of distribution lists and contribution lists. They have a role in modeling the external interfaces to a system, to explain the source of input and the destination of output.

When a model like this is considered in a larger context of network security, it becomes important to have a more precise notion how input and output are handled. For this reason, we will now give more detail on how users may be incorporated formally into the model.

Let us say that users are actually special subjects. They are exceptional because they cannot have execute accesses, and because they cannot have both read and write accesses. This split between a user as an output sink and as an input source reflects the lack of a deterministic circuit between the outputs to a user and its subsequent inputs to the system. A human operator would be modeled as a pair of users (eyes and hands respectively). A full duplex network connection is a pair of simplex connections.

Naturally, the distribution list of an output user consists only of that user; the contribution list of an input user is also just that user. The distribution list of an input user and the contribution list of an output user are, by convention, all-inclusive, so as not to interfere with users' source and sink roles.

Users are also exceptional in that they cannot directly request changes in the system state, such as access or level changes. Requests of this kind do not really come from users; they come from processes running software that has interpreted the user's keystrokes. This is why we cannot say, for example, that any user, as a subject, has the privilege to change the individual access components of any object it owns.

INFORMATION FLOW AND ACCESS

The comment is often heard that "read" and "write" access are meaningless terms in security models; one could exchange them systematically, or rename them "Brenda" and "Charlie", and the resulting model is logically equivalent to the first. Behind the comment is the fear that one could misinterpret the model and implement an insecure system.

Another problem with formal models is that, if they are at all complex, their consequences are not obvious in terms understandable to the user community. They

may even be internally inconsistent.

In partial response to these concerns, one can prove theorems about the model. For example, the Criteria document requires that a model be "proven consistent with its axioms," and it mentions in the Glossary, as part of the entry for "Formal Security Policy Model", that the transitions of an acceptable model must be proved to preserve the security of system states. These requirements address the potential internal inconsistency of a model possessing both general axioms and specific rules.

There are other kinds of general properties one might wish to prove about a formal security policy model. If the model has capabilities, i.e., access tickets that can be passed from one subject to another, one might ask whether it is possible to determine which subjects could eventually receive a particular capability originally in the possession of some given subject. A similar question could be asked about the propagation of privileges in the A1 model above.

One might also wish to prove that information cannot be compromised. That is, there should be no "information flow" from a high-level source to a lower-level destination. The simplest formal expression of this requirement is given in an SRI model [FLR77]. Looking only at the inputs and outputs of a system, it says that lower-level outputs do not depend on higher-level inputs. If all inputs that are not below a given level were eliminated from a system history, the outputs at that level should be unchanged. This is an example of a non-interference assertion, and it is referred to as the multilevel security property [GoM84]. An attempt to prove the multilevel security property for a system is referred to as information flow analysis.

Information flow analysis is usually applied only to formal specifications or programs, in an effort to detect covert channels. It is contended that there are at least some models that could also benefit from information flow analysis. It seems likely, for example, that the change-subject-current-security-level channel could have been caught this way.

For an access control model, that is, one in which subjects have read and write access to objects, information flow analysis clarifies the meaning of the access modes. Write access means that the data content of an object can change; read access means that changes in another object can depend on the data content of one read.

Multilevel Security for the A1 Model

The formal statement of the A1 model, with the appropriate information flow assumptions, and the proof of a multilevel security property, are beyond the scope of this paper. At the time of writing this paper, the multilevel security property for the A1 model has been proved under some simplifying assumptions, e.g., that no privileges are invoked and that no changes occur in subject or object security levels.

The information flow analysis affected the design of the model by adding several new ingredients, such as object values, but it also had the retroactive effect of forcing the inclusion of the Creation and Access Change conditions for secure transitions. These latter additions are expressed without reference to the new ingredients, so they could be retained in a simplified version of the model from which those new ingredients had been deleted.

CONCLUSIONS

The model described in this paper is too complex to qualify as a "minimum A1 policy model", but it embodies some suggestions about what is needed in models and leads to some conclusions about what is still missing.

A minimum A1 policy model could be obtained from the one given, by leaving out all security conditions except Read, Write, and Tranquility; and dropping the integrity components of the security level. The result would still have an unusually restrictive interpretation of discretionary security.

The requirements for proving the model "consistent with its axioms", and for proving that state security is preserved, are not believed to be relevant for models not having specific transition rules. Specific transition rules were omitted because they would restrict the applicability of the model unnecessarily to an even smaller class of TCB's than that implied by the access control approach. The design guidance afforded by specific rules can be left to the formal specification of a system.

What modeling needs is the formal expression of high-level policies, which can be used to evaluate concrete models aimed at more specific policies, like that for A1-class TCB's, or military message systems [MLH84]. The simple requirement for internal consistency is a good example. Another is the SRI multilevel security property, though it is not quite right because it does not reflect various privileged actions, such as downgrading by appropriate authorization, that are permissible under DoD policy.

It is hoped that the ideas presented here for handling individual access control and privilege will be useful in future modeling efforts. At the same time, it is important to push forward in the development of new models in areas not covered by the Criteria: applications, new system architectures, and networks.

REFERENCES

[Bib77] K.J. Biba, "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, The MITRE Corporation, Bedford, MA, April 1977.

[BLP75] D.E. Bell and L.J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," ESD-TR-75-306, The MITRE Corporation, Bedford, MA, July, 1975.

[CSC83] "Department of Defense Trusted System Evaluation Criteria," CSC-STD-001-83, 15 August 1983.

[GoM84] J.A. Goguen and J. Meseguer, "Unwinding and Inference Control," Proceedings of the 1984 Symposium on Security and Privacy, 84CH2013-1, IEEE Computer Society, pp. 75-87.

[Gro76] M.J. Grohn, "A Model of a Protected Data Management System," I.P. Sharp Associates, Ltd., Ottawa, Canada, June, 1976.

[MLH84] J. McLean, C.E. Landwehr, and C.L. Heitmeyer, "A Formal Statement of the MMS Security Model," Proceedings of the 1984 Symposium on Security and Privacy, 84CH2013-1, IEEE Computer Society, pp. 188-194.

An Overview of the Kernelized Secure Operating System (KSOS)

Tom Perrine
John Codd
Brian Hardy

Logicon - Operating Systems Division

INTRODUCTION

This paper will present the Kernelized Secure Operating System (KSOS) as it exists today, with emphasis on its security policy, architecture, and its ability to support secure applications, specifically the ACCAT GUARD multi-level-secure application. A discussion of plans for its future development and qualitative performance information are also included.

Description of KSOS

KSOS is a multi-level-secure (MLS) computer operating system consisting of a security kernel, Non-Kernel Security-Related (NKSR) utility programs, and an optional UNIX application support environment. A KSOS software development environment will also be provided.

KSOS was designed to be a provably secure replacement for the UNIX operating system, Version 6. The system runs on an unmodified Digital Equipment Corporation PDP-11/70. The KSOS system enforces a formally specified security policy, encompassing mandatory access, integrity, and discretionary access models. Application programs that have been developed for the non-secure UNIX operating system may be ported to the highly-secure KSOS environment with minimal effort.

The KSOS system has been informally evaluated by the DoD Computer Security Center (CSC) and has been characterized as "an excellent base for developing into an A1 system."[1] This is especially significant, as KSOS was designed many years before the DoD CSC Trusted Computing System Evaluation Criteria [2] was published.

KSOS Functional Architecture

The KSOS system is made up of the following functional areas: the Kernel, the Non-Kernel-Security-Related (NKSR) software, and the Kernel Interface Package (KIP). The security kernel provides the basic operating system functions of the system and enforces the security policy. The NKSR programs provide additional operating system functions and utility operations. The KIP is a UNIX-compatible run-time environment for running UNIX application programs. In the future, KSOS will include a fourth area: the KSOS development environment.

This work was sponsored by Naval Electronics System Command (NAVELEX) contract number N00039-83-0144. UNIX is a trademark of AT&T Bell Laboratories. DEC, PDP, and VAX are trademarks of Digital Equipment Corporation.

History of the KSOS project

KSOS was originally intended to be the first production-quality multi-level-secure operating system and to provide a secure UNIX replacement.[3] It was based on the results of the UCLA Data Secure UNIX [4] and MITRE security kernel experiments. [5][6]

The KSOS project began in 1977 at Ford Aerospace and Communications Corporation (FACC). Since 1981, Logicon has continued to support and develop KSOS.

While KSOS was being developed to prove the concept of a buildable security kernel, the Advanced Command and Control Architectural Testbed (ACCAT) GUARD application was developed to prove the concept of a multi-level-secure "guard" program to provide a verifiably secure access to multi-level all-source databases distributed on ARPANET-like networks. [7] Initially, a UNIX prototype of ACCAT GUARD was developed in anticipation of a secure UNIX replacement, to serve as the base operating system for GUARD.

The KSOS Kernel Interface Package (KIP) was developed to allow the migration of the UNIX-based GUARD prototype to KSOS with minimal software changes. This package has since been used to port other UNIX software to KSOS, and has demonstrated significant performance improvement over the previous UNIX compatibility supported by the original KSOS UNIX Emulator.

ACCAT GUARD, using the KIP, is currently undergoing accreditation review as a multi-level-secure system. It has shown that KSOS is robust and capable of supporting a rigorous Security Test and Evaluation (ST&E).

KSOS also served as a testbed for many (then advanced) features of secure systems. KSOS has helped to prove many of the concepts embodied in the DoD Trusted Computer Evaluation Criteria.

KSOS SECURITY POLICY

The KSOS security policy encompasses three orthogonal policy models, one for mandatory security, one for integrity and one for discretionary access protection. These models are defined in terms of objects (data containers) and subjects (processes acting on behalf of a user) and the rules under which subjects may access objects. If any of the three models would deny access to an object, the access is denied, i.e. all of the assertions of all of the models must be maintained at all times. All accesses to objects are mediated by the Kernel, ensuring mandatory access controls. It is not possible for a subject to access an object without Kernel intervention.

Every object in the system is marked with trusted labels, which consist of a security level, an integrity level and discretionary access permissions. Every object is labeled (marked) by the Kernel when it is created. These labels are used by the implementations of the security policy, within the Kernel, to permit or prevent accesses, as specified below.

Mandatory Security Model

The KSOS mandatory security model is the Bell-LaPadula model [8], which is in turn based on Department of Defense policies for the handling and dissemination of classified material. It is described in terms of the "simple security property,"

which determines what information a user may see, and the "security *-property," which prevents a user from lowering the classification of information (downgrading). The KSOS model includes both security classification levels, such as "SECRET" or "TOP SECRET", and "need-to-know" categories, such as "NO FOREIGN" or "NO CONTRACTOR".

Integrity Model

In this model, integrity is defined as the mathematical dual of security, and the intent of the model is to protect the system's information from modification, while allowing it to be read by any process.

As the mandatory security model controls who may obtain data stored in the system, the integrity model controls who may place data into the system, and how that data may be combined with other data.

There are two integrity properties which control object accesses: the "simple integrity property" and the "integrity *-property". The simple integrity property prevents a high-integrity process from reading low integrity data, which might then be written into a high integrity object. [9] The integrity *-property prevents a low integrity process from writing into a high integrity object. These properties keep low integrity ("less trusted") information from propagating into high integrity ("more trusted") objects.

An example is the KSOS mount table. Every process in the system may need to read this database, but only the operator or system administrator may change it. This database is assigned an integrity level of OPERATOR, which prevents user processes (running at the lower integrity level USER) from writing to the database file.

Discretionary Access Model

The KSOS system also includes a discretionary access model, which is derived from the UNIX discretionary model. [10] As in UNIX, every person using the system is assigned a user identifier, and every user is a member of at least one group of users. Every object has a permission set for the objects' owner, other members of the owner's group, and other users of the system. There are three permissions; "read", "write" and "execute", indicating the ability to extract information from, send information to, and (for files) the permission to load the file into memory as an executable program. These permissions are established at the discretion of the objects' owner.

KSOS KERNEL

The heart of the system is the KSOS Security Kernel. The Kernel is a complete operating system which provides a secure environment for the execution of user programs. It supports multiple isolated processes, a file system and a set of "supervisor" services. The Kernel is based on a reference monitor concept, wherein every access to every object is mediated by the Kernel according to its security policy.

Kernel objects and subsystems

The Kernel supports the following objects: processes, memory segments, devices, disk extents, files, and file subtypes. These objects are created and destroyed only by the Kernel, and all accesses of the objects are controlled by the Kernel in accordance with its security policy. Each object is the responsiblity of one of the major subsystems of the Kernel, described below. All Kernel objects are labeled with their security and integrity levels and discretionary access information. All Kernel objects are assigned a unique identifier called a Secure Entity IDentifier (SEID, pronounced "seed") which is a binary quantity. The SEID can be thought of as the Kernel's "name" for an object. A SEID is not a capability, and having the SEID does not imply any access privileges to the object. The only way to manipulate the Kernel objects is through the use of Kernel calls, identifying the object by its SEID.

The KSOS Kernel is split into four major functional areas: Process Management, Memory Management, Input/Output, and the Reference Monitor. Each of these areas are responsible for maintaining internal Kernel databases reflecting the state of all objects under the control of the Kernel.

Processes

All KSOS processes are managed by the Kernel Process Management Subsystem. This subsystem creates and deletes processes from the system, schedules them for execution and controls all interprocess communication. The real-time clock is also implemented in this subsystem, as are pseudo-interrupts and software trap handlers. Some of these sub-subsystems are visible to a user process by means of Kernel calls, others, such as the scheduler, are acting "behind the scenes," and are invisible to the user process.

The Kernel process is the only active object (subject) in the KSOS object space. Processes are the means by which programs are executed on the machine. A process performs its work by manipulating KSOS objects, i.e. reading and writing to and from files or devices, or communicating with other processes. A process consists of a program image, process context information, a memory address space, and a processor state.

New processes come into being at the request of other processes through use of Kernel calls.

A process may execute with special privileges. Such a process is a "trusted" process and may violate the KSOS Kernel security policy or use system control functions. These processes can become privileged only through the actions of the System Administrator.

Memory Segments

The Kernel Segment Management Subsystem is responsible for allocating, deallocating, swapping and controlling access to the segments of the primary memory of the system. The details of physical memory management and swapping are typical of many operating systems, and will not be discussed here. We will concentrate on the novel features of the KSOS segmentation subsystem.

Memory segments are an abstraction of the virtual memory visible to a process. Memory is made up of segments, each of which resides in either the Kernel, Supervisor, or User domains. (The domain structure is provided by the PDP-11 memory management hardware.) The KSOS Kernel resides in Kernel domain, NKSR programs typically reside in Supervisor domain and user application programs reside in User domain. A process program image and its data reside in a single domain, but transfers of control may span domain boundaries (under Kernel supervision).

User memory is organized into named segments. Like processes, segments are named by their SEIDs. A user process can have up to 16 segments resident in memory at any time, eight of which are allocated by the hardware architecture for instructions (I-Space) and eight of which can contain only data (D-Space). Each segment is limited in size to 8K bytes. Therefore, there is a limitation of 64K bytes of instructions and 64K bytes of data per memory management domain, per process. The maximum of 64K bytes address space can be spanned by the memory management system only if every segment is of the maximum size.

The Kernel permits a process to manage its data segments in a manner that can be used to best fit the application. For example, a process can dynamically create and destroy data segments, as well as determine which of its known segments are to be resident in main memory, and where, at any given time.

One of the more novel (and useful) features of the KSOS segmentation subsystem involves the use of shared segments. A process may create a segment, specifying that it is to be "sharable". Any other process may then "rendezvous" with the segment, under Kernel mediation. At this point, both processes have the same physical memory mapped into their virtual address spaces. This feature permits a very high bandwidth communication path for cooperating processes.

Input/Output Management

The KSOS I/O Subsystem is responsible for managing devices, disk extents, files and file subtypes. Devices, disk extents and files are increasingly abstract representations of physical input/output devices. File subtypes provide an extension of the Kernel defined object types.

Devices

Devices under KSOS are handled by the low-level device drivers within the Kernel I/O Management Subsystem. It is at this level that interrupts are handled, device commands and data are sent to and from the devices, and data buffers and device status registers are examined. Storage device contents are addressed by logical block number. Non-kernel programs are unable to perform I/O directly, but must make requests to the Kernel to have it perform I/O on their behalf.

Devices have minimum and maximum security levels that indicate what classifications of data may be sent to or received from it.

User terminal devices are handled in a novel fashion. There are several "virtual paths" to each terminal, each of which can be at a different security/integrity level. One of these paths is reserved for use by the system and is called the "Secure Path." This path provides a trusted communications path from the user to the Kernel, for use in invoking trusted functions. When the user uses the "secure attention" key, it is guaranteed that he is communicating with trusted software, and not a program that may have been left executing at the terminal by

another user. This secure path is used during login, logout and any time that the user must be communicating with trusted NKSR software.

Disk Extents

Disk extents are the next higher level of abstraction of devices, specifically mass-storage devices such as disks. An extent is a named set of contiguous blocks on a given disk device, which can be used as a private, logical storage device. As a "device" (and an object), an extent has security and integrity information governing information flows to and from the extent. The contents are selected by relative block number from the beginning of the extent. Disk extents are intended for use by programs that wish to manage their own storage space, without the imposition of any file structure by the Kernel. They might be used, for example, by a relational database, which uses its own special internal "file" format on top of an extent.

Files

The KSOS Kernel provides a "flat" file system. There are no directories or links and files are named only by their SEIDs. All of the security and integrity information is checked and maintained at the Kernel level. A file system resides in an extent on a disk, and may be "mounted" (made known to the system) or "unmounted". Files are allocated in 512 byte blocks, the blocks of which need not be contiguous. Both random and sequential access are supported. As files are created, they are marked by the Kernel with the security and integrity levels of the process that created it. Files may be opened, closed, read or written only by making requests to the Kernel.

File Subtypes

File subtypes allow a System Administrator to define a special type, or flavor, of file for special handling by the system. These are called "file subtypes," and may be thought of as a private object. They can be used in support of object-oriented programming, to implement special-use reference monitors on top of the Kernel.

File subtypes are a special object in KSOS. They have security, integrity and discretionary access information, just like other objects. They also have an owner. In practice, the discretionary access allows write access to the owner only. This becomes a "private type" of the type manager (the owner). Only the owner may open the subtype for writing.

When a file is created, a subtype may be specified. At this point, the subtype is entered into the Kernel's information about the file. Later, when a process attempts to open the file, it must have already successfully "opened" the subtype with the same mode, or the file open will fail. As the owner is the only process which may open the subtype for writing, only the owner may open the "subtyped" file for writing.

For example, the UNIX Directory Manager (UDM) implements the hierarchical UNIX-like file system from the more primitive Kernel file system. The UDM creates Kernel files with a subtype that only it may write. These files are then used by UDM as UNIX directories.

Later, any process that attempts to open the directory file, for writing, must have already opened the subtype for writing. As the UDM is the only process which may open the subtype for writing, no other process may open the directory file for writing. This ensures the integrity of the information in the directory file. However, any process may open the directory file for reading (subject to other access constraints, of course). Subtyped files can be thought of as a "non-discretionary, discretionary access model", where there are permissions for reading, writing and executing, but the permissions are set and maintained by the Kernel, and may not be changed at the discretion of a user program.

This feature of KSOS is very useful for implementing special-purpose databases where only a single process which "owns" the database is to be allowed to update it. It has also been used for the TCP/IP network daemon, to protect files used by the network manager, and will be used to implement multi-level-secure mailboxes for the Secure Mail Facility.

Reference Monitor

The KSOS reference monitor has been mentioned in passing in the discussions of the other Kernel functional areas. The reference monitor ensures that all accesses to the objects protected by the Kernel are permissible under the KSOS Security Model. This module is invoked by the other functional areas to determine the validity of the attempted accesses (or information flows).

NON-KERNEL SECURITY-RELATED (NKSR) SOFTWARE

The NKSR software that is part of the KSOS system falls into four functional areas: Secure User Services, System Operation Services, System Maintenance Services, and System Administration Services.

Secure User Services

The Secure User Services NKSR programs are responsible for initializing the KSOS system and providing a secure path from the user to all of the trusted NKSR services. Programs in this area include:

* Initial Process

  This program is responsible for initializing the security levels of the KSOS system objects. It is the first process to execute after the bootstrap process.

* Secure Server Process

  This is the command processor of the system. It manages the different virtual paths to the user's terminal and invokes other NKSR services at the request of the user.

* Login and Logout

  Login is responsible for performing user authentication functions and creating the initial user environment. Logout destroys the user's process and makes the terminal available to other users.

System Operation Services

These programs contain functions that are necessary to support a general purpose operating system. Such functions include:

* Line Printer Daemon

  This is the "daemon" process that performs line printer spooling.

* Mount/Unmount

  These facilities control the mounting and unmounting of file systems.

* Network Daemon

  This daemon process handles the TCP/IP DDN or ARPANET connections.

* UNIX Directory Manager (UDM)

  UDM implements a hierarchical, UNIX-like file system from the more primitive Kernel "flat" file system. Text string names and directories are implemented by this program.

System Maintenance Services

These programs provide the necessary functions to maintain the KSOS file systems in a usable state, such as:

* Storage Consistency Check (STC)

  STC checks the consistency of the Kernel file system, reporting any lost blocks, duplicated blocks, etc.

* Directory Consistency Check (DCC)

  DCC checks the consistency of the UNIX file system maintained by the UNIX Directory Manager, reporting directories that are in an inconsistent state, etc.

* File System Dump/Restore

  These utilities provide backup and restore of KSOS file systems.

System Administration Services

This class of programs provides the functions needed to assist the System Administrator in easily managing a multi-user, multi-level system. These functions include:

* User Registration and Removal

  This program allows the System Administrator to add new users, remove users and identify the clearances of the users to the system.

* System Profile Maintenance

  This program maintains the system profile database, which describes the particular KSOS installation in terms of software versions, site name, etc.

* Audit Capture Process (ACP)

  The Kernel and NKSR software generate audit events for several reasons, including user login, logout, object creation, access failures, activity on possible covert channels, etc. The Audit Capture Process receives these messages and writes them to an audit file.

KERNEL INTERFACE PACKAGE (KIP)

The Kernel Interface Package (KIP) provides a UNIX Version 6 system-call compatible interface, except for those system calls which have been identified as security flaws of UNIX. (The functions of the latter system calls have been subsumed into the NKSR software.)

The KIP is a library of subroutines and functions, one for every supported UNIX system call, which are linked with the user program. These library functions invoke KSOS Kernel calls to carry out their functions. Very little data is maintained by the KIP from call to call. The KIP can be viewed as a UNIX to KSOS call translator.

The KSOS KIP allows the easy migration of existing software written for the UNIX environment to the multi-level-secure environment of KSOS. This method of providing a UNIX environment allows source-level compatibility, and better performance than the original UNIX Emulator.

KSOS DEVELOPMENT ENVIRONMENT

At the present time, the KSOS software development environment requires a PWB/UNIX system. All programs are prepared using the UNIX development tools.

There are plans to provide a full KSOS development environment running on KSOS. This will give the KSOS system the ability to maintain itself. Initially, a minimum set of software development tools will be installed on KSOS. As a preliminary feasibility study, the "ed" editor was ported with very few changes, making use of the KIP.

The next phase is to select the set of tools that will be ported from UNIX. Some candidates for UNIX software to be ported are the UNIX "shell", the "C" compiler, the UNIX assembler, the loader and the Source Code Control System. A screen editor will also be chosen and ported.

ACCAT GUARD on KSOS

The Advanced Command and Control Architectural Testbed (ACCAT) GUARD application is a multi-level-secure application developed for the Naval Electronic Systems Command (NAVELEX).

ACCAT GUARD will provide a certifiably secure interface between two computers or subnets on the Defense Data Network (DDN) which are operating at different security levels.

The GUARD system is responsible for secure exchange of information between HIGH level and LOW level network connections. The boundary between these HIGH and LOW levels is guaranteed through the KSOS multi-level security protection mechanisms in accordance with the DoD security policies.

ACCAT GUARD allows the passing of information from the LOW to the HIGH network automatically, but ensures that all information passing from the HIGH network to the LOW network is subjected to manual sanitization and manual review for downgrade. The downgrade is performed by a formally specified, trusted program, the Downgrade Trusted Process (DGTP), which is the only component of the GUARD application software which is trusted (or privileged) to perform the downgrade (by the KSOS security mechanisms).

The ACCAT GUARD system has passed Security Test and Evaluation (ST&E), and is under accreditation review by the Defense Intelligence Agency. The ST&E has shown the robustness of the Kernel and the application, by executing under a wide variety of load conditions for extended periods of time. Most importantly, no security weaknesses were discovered during the ST&E.

ACCAT GUARD was developed with several goals in mind. Its primary goal is to validate the concept of a guard as a buildable multi-level-secure application. But in addition it validates the concept of KSOS as a production-quality security kernel, and demonstrates the capability of KSOS to host an application which was originally written for execution on UNIX, using the KIP.

KSOS - FUTURE PLANS

KSOS development is continuing at Logicon, in support of the ACCAT GUARD system. Additional areas of research include developing KSOS into a DoD CSC A1 certifiable system, porting KSOS to alternate hardware architectures, and providing performance and functional enhancements to the existing system.

DOD CSC A1 Certification

KSOS was designed to be a secure system before the DoD CSC published the Trusted Computer System Evaluation Criteria. The security policy was not an add-on, and security was the prime design goal. The KSOS Kernel is described by a Formal Top Level Specification (FTLS), expressed in SPECIAL. An informal review of correspondence between the FTLS and the KSOS implementation has been performed, but the formal document has not yet been produced. KSOS has been characterized as an excellent base from which to build a secure system.

Over the lifetime of KSOS, several verification efforts have been performed by the MITRE corporation. The results of these efforts are available from MITRE. The most recent effort [11] involved the Kernel FTLS, which was examined using the

Hierarchical Design Methodology (HDM) [12] tools.

This effort produced 1638 theorems, 939 of which were proven trivially and 431 of which were eliminated as duplicates. This left 268 theorems, of which the theorem prover was able to prove 47, leaving 221 unproven theorems. The number of unproven theorems can be further reduced to a minimum by several methods, including adding additional assertions to the specification. Any remaining unproven theorems which are shown to indicate covert channels may be handled by limiting the bandwidth of the channel, or auditing the use of the channels.

Since this verification effort, however, the Kernel has had minor changes. The current FTLS correctly reflects the state of the KSOS implementation, but needs additional work in the area of specifying more assertions, to allow more of the theorem proving to be performed automatically.

Although Logicon is not currently under contract to develop KSOS to the A1 level, we expect this effort to proceed in parallel with further development of the system, i.e. all changes that are made will be designed with eventual A1 certification in mind. According to the Computer Security Center, "KSOS is an excellent base for developing into an A1 system." All of the areas in which KSOS is deficient have been identified, and the necessary development activities have been specified.

New hardware architectures

One of the original design goals of KSOS was to provide an easily portable system that could be moved across machine architectures with minimal re-design effort.

We are currently investigating the migration of the KSOS system to other hardware architectures, specifically, the Digital Equipment Corporation VAX. Most limits of KSOS performance are imposed by the architecture of the PDP-11. Migration to a VAX will provide many benefits, especially allowing KSOS to reside on a wide price/performance range of machines, all running functionally identical Kernel software, with only minor changes to support different central processors.

It is expected that we will be able to move KSOS to the new PDP-11/73 hardware which is a less-expensive, single-board implementation of the full PDP-11 architecture. This could be accomplished with little or no effort providing a low-cost hardware base for the current PDP-11 kernel.

Other architectures which may be examined in the future include the Motorola 68000 family, National Semiconductor NS32000 series and other high-performance microprocessor families.

Performance and Functional Enhancements

Several opportunities for performance and functional improvements have been identified within the KSOS system. In particular, there are plans to improve the Kernel Input/Output Subsystem, specifically in the areas of terminal handling, asynchronous I/O and device request handling.

The current KSOS design allows multiple processes to share a single copy of read/execute-only instruction spaces. However, the implementation of shared instruction segments is incomplete. Shared instruction space will also decrease the swapping load on the system and increase throughput dramatically.

The current KSOS KIP supports a UNIX Version 6 environment within the MLS environment of KSOS. Other system call translators will be written for other UNIX versions. For example, interface packages could be written to support the 4.2BSD or AT&T System V system call interfaces. Once the system calls are available, application software can be ported with minimal effort.

PERFORMANCE

This section reports the results of some informal, qualitative performance measurements which were performed recently. The figures are not intended to state absolutely the performance differences between the environments, but to give a feel for the performance of a security kernel and point out that applications that are intended to run on a kernel will benefit if written to use the native environment of the kernel. Applications can be ported directly, using the KIP, but performance will be traded for ease of migration.

These programs use the following environments: the KSOS Kernel native-mode run-time environment, the UNIX Version 6 environment as provided by the Kernel Interface Package on top of the Kernel, and the PWB/UNIX environment. All systems were run on the same PDP-11/70 at Logicon.

Tasks

The following types of programs were identified as being "interesting", because they exercise various parts of the different environments and are typical of the tasks of application programs.

* CPU intensive

  This task is to repeatedly compute the prime numbers less than 10000, using a relatively inefficient algorithm (Sieve of Eratosthenes).

* Interprocess Communication (small messages)

  This task is to pass many 10-byte messages from one process to another.

* Interprocess Communication (large message)

  This task is to repeatedly pass a single large (1000-byte) message from one process to another and back again.

* File Input/Output Intensive

  This task is to open an existing file, write 64 blocks of data, rewind the file and read the data.

* Process Creation

  This task times the various process creation mechanisms. A process is loaded into the system. This process starts off a child process, and then exits. The child does the same thing. This continues for 100 process creations.

Experimental Results

The selected test cases have been chosen to correspond to the types of operations typically performed by application software. Test programs were written in C for execution under UNIX. These UNIX-based programs were executed under KSOS/KIP environment with no changes to the software. Finally, each program was modified to execute directly with the KSOS Kernel. For each test case, the total elapsed time required to complete the test was measured. The test results were normalized with respect to the time required to complete the same functional test under the UNIX operating system. These results are summarized in the following table.

Performance Characteristics Ratio - KSOS vs UNIX

Test Scenario	UNIX	KSOS (KIP)	KSOS (Kernel)
CPU-bound	1.0	1.0	1.0
IPC (10 bytes/msg)	1.0	25.0	3.9
IPC (1000 bytes/msg)	1.0	65.0	1.7
IPC (5000 bytes/msg)	1.0	325.0	0.3
I/O-bound	1.0	1.6	1.4
File Creation	1.0	139.0	13.9
Process Creation	1.0	67.6	30.8

Software portability versus desired performance characteristics continues to be the topic of intense debate and trade-off analyses. This issue is especially significant when developing application software which will operate under a secure system such as KSOS. As shown in the table, rather significant gains in performance can be achieved by tailoring the application to the features provided by the kernel. In particular, the ACCAT GUARD system performance was improved by approximately a factor of three by applying this concept to a small set of carefully chosen application modules.

It is interesting to note that for CPU-bound application software, the KSOS Security Kernel does not impose any performance penalties when compared to UNIX. As for the CPU-bound software, I/O-bound software only has a marginal decrease in performance. This result was anticipated due to the differences in the I/O design philosophy between UNIX and KSOS. Furthermore, the version of the test that executed directly under the KSOS Kernel was only slightly improved over the performance of the similar test which executed under the KSOS/KIP environment. Therefore, for this case of application processing, the UNIX-based software can be migrated to KSOS with relatively small performance penalties.

In applications which required a high degree of interprocess communication, the tests clearly indicate that using the features of the kernel will provide rather significant increases in performance, particularly when rather large messages must be exchanged. It is interesting to note that as the message size inceases, the interprocess communication features provided by the KSOS Kernel will permit a higher effective throughput rate than UNIX.

Finally, new object creations, particularly processes and UNIX file systems managed outside the KSOS Kernel, will require considerably more computational resources than non-security kernel-based operating systems such as UNIX. Again, this result was anticipated due to significant differences in the design between UNIX and KSOS.

CONCLUSION

KSOS is a demonstrated, full-featured operating system built according to the latest philosophies in computer security. It runs on commercially available hardware and holds the promise of providing secure processing on a family of hardware that spans the spectrum of computers from micro to mainframe. With KSOS, a system implementer may easily port existing operational software to a secure environment and not necessarily pay a great performance penalty. KSOS is an ongoing software project that offers a solution to the MLS problem while continuing to improve its features and performance.

REFERENCES

[1] Letter to Commander, NAVELEX, 25 June 1984, subject: KSOS-11 Security Assessment

[2] CSC-STD-001-83, "Department of Defense Trusted Computer System Evaluation Criteria," 15 August 1983.

[3] E.J. McCauley and P.J. Drongowski, "KSOS: The Design of a Secure Operating System," in Proceedings, AFIPS National Computer Conference, AFIPS Press, Arlington, Va., 1979, Vol 48, pp. 345-353.

[4] G.J. Popek, M. Kampe, C.S. Kline, A. Stoughton, M. Urban, and E.J. Walton, "UCLA Secure UNIX," in Proceedings, AFIPS National Computer Conference, AFIPS Press, Arlington, Va., Vol 48, pp. 355-364.

[5] W.L. Schiller, "Design of a Security Kernel for the PDP-11/45," MITRE MTR-2709, MITRE Corp., Bedford, Mass., June 1973.

[6] K. Biba, J. Woodward, and G. Nibaldi, "A Kernel Based Secure UNIX Design," MITRE ESD-TR-79-134, MITRE Corp., Bedford, Mass., May 1979

[7] D. Baldauf, "ACCAT GUARD Overview," MITRE MTR-3861, MITRE Corp., Bedford, Mass., Nov. 1979

[8] D.E. Bell and L.J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," MITRE MTR-2997, MITRE Corp, Bedford, Mass., July 1975.

[9] Biba, K.J. "Integrity Considerations for Secure Computer Systems," MITRE MTR-3153, MITRE Corp., Bedford, Mass., June 1975.

[10] D.M. Richie and K. Thompson, "The UNIX Timesharing System," in Communications of the ACM, Vol. 17, No. 5, pp. 365-375. (May 1974)

[11] K.E. Kirkpatrick, "KSOS Verification Part I: Analysis of the Specifications and Use of the Verification Tools," (working paper), MITRE Corp., Bedford, Mass., February 1982

[12] B.A. Silverberg, "The HDM Handbook, Volume II: The languages and tools of HDM," SRI International, 1979.

FUTURE DIRECTIONS OF SECURITY FOR SPERRY SERIES 1100 COMPUTERS

T.M.P. Lee

Program Manager, Systems Security
Sperry Corporation
Computer Systems
Roseville, Mn.

ABSTRACT

At the third of these seminars -- four years ago -- we discussed the evolution of computer security features in our past and present products. With the first delivery this last April of the large-scale 1100/90 computer, which includes a new addressing and protection architecture especially enhanced for security and integrity, we can now discuss the direction the Series 1100 Operating System will be taking over the next half-decade or so. As in the past many improvements will be made in direct response to specific marketplace requests -- in this case, both individual customer recommendations and the various steps needed to progress up the levels of the DoD Trusted Computer System Evaluation Criteria.

INTRODUCTION

I am pleased to be able to make an announcement. Two months ago Sperry computer systems began a multi-million dollar research and development program in computer security. This program will result in OS/1100, the operating system for our Series 1100 family of computers, being certifiable at the B1 level in early 1987, and B2 as quickly as possible thereafter.

The purpose of this report is to describe the changes and enhancements we will be making to the system as part of this program. To a large extent the report has been prepared for an audience not familiar with our systems and terminology; many details of interest only to people such as system analysts (or our evaluation team) will not be covered. It does cover four topics:

1) the functional software changes to meet B1 requirements
2) a number of additional changes recommended by our users
3) the new hardware architecture just delivered in our 1100/90 that will make it possible to do a B2 and B3-level restructuring without suffering unacceptable performance penalties
4) how we intend to use that architecture.

Those of you familiar with other systems will recognize a number of the features discussed here as similar to those in other systems or in the literature. This report will not attempt to make a comparative analysis, but we do of course freely acknowledge our debt to the

accumulated wisdom of others.

The two-year estimate for attaining B1 is real, for the three most important reasons -- we know fairly accurately from a detailed technical plan how many man-years of work it is going to take, we have the funding in place to get that much work done by that date, and we have just about completed getting the right number of people in place to do the work.

To further emphasize our commitment to trusted computer systems, and our faith in the underlying approach, as soon as a certified B1-level system is released we plan to institute a new policy towards the handling of user reports dealing with potential security defects. Any properly documented and reported design or implementation flaw that permits a credible breach of security will be handled at the highest possible level of priority within our software maintenance organization. For those familiar with our procedures, this means that a fix to eliminate, repair, or render the flaw unexploitable will be developed in response to a Priority 4 SUR (Software User Report.)

During the interval of responding to the flaw report, knowledge of the flaw will be carefully controlled and limited. It might be appropriate, in fact, to notify all users as quickly as possible of the existence of the flaw, even before we have a fix for it, but we need to think that through a bit more. At the least we would certainly notify the DoD Computer Security Center, and would hope it would reciprocate by telling us about the vulnerabilities reported to it through its procedures.

CURRENT IMPROVEMENTS

We have to walk before we can run. Those of you who have been to several of these seminars before, and certainly that includes our competitors, are well-aware, even though it has never been publicly announced, that we have been involved in what used to be called an "informal evaluation" of OS/1100 by the DoD Computer Security Center. The draft report of that evaluation carried no surprises -- anyone who knows our system can compare it against the Orange Book -- but it did make mutually clear a few things that needed to be done to reach level C2. Our user's meetings have also recommended to us a number of other desirable enhancements in roughly the C2 area, most of which will be implemented.

The main thing missing from the C2 requirements was an approach to discretionary access controls that clearly met the spirit of the Orange Book. What had been in our system could have been successfully argued as having met the requirements, but it would have been a tour de force. Software is now being integrated into the system that associates with each file (and eventually other classes of protected objects) the identity of its owner; previously the closest notion of ownership we had was that of the project or account the file belonged to. The owner of a file will have the ability to explicitly give a list, which he, and only he, can later change, of which other users are able to access that file, and in which ways. If no list is given, the file can only be accessed by its owner. (By the way -- these

access control lists are named entities and thus the same list can be easily attached to several files at once.)

Another major change necessary for even C1 level accreditation arises in our transaction processing system. For many years OS/1100 has supported what one might think of as a special-purpose high-performance sub-operating system designed primarily to handle applications like airline reservations or customer inquiries. This sub-operating system (called TIP, for Transaction Interface Processor) does not fit the computer science model of one process per user, but in fact much better fits the model of one process per kind of transaction. The TIP system itself has no notion of the identity of a user, although application programs under TIP may choose to identify and authenticate their users.

It is clear that for a clean accreditation at the C levels, and even more so at the B levels, this way of doing business is unsatisfactory, since it forces part of the TCB to be in applications programs. Accordingly we have in the works a number of changes to ensure that all users must logon (uniquely identify and authenticate themselves) in an effective and uniform manner, no matter what kind of application (time-sharing, transaction, or real-time) they are connecting to. This ensures that any security-relevant activity -- recorded in an audit trail -- can be traced to the paricular person responsible. Other changes in the TIP environment will also occur as part of the level B1 enhancements.

B1 ENHANCEMENTS

Most of what we have to do to meet the level B1 requirements can be easily stated: files and other protected objects need to have security levels and compartments associated with them, and hard-copy output needs to be labelled with human-readable security labels. Our present mechanisms, which have a notion of security level and a kind of compartmentation, will be modified to conform strictly to the lattice policy outlined in the Orange Book and elsewhere. At this time we are not sure about how many compartments to support; a number in the range of 32 to 64 can be done easily, but we also have under consideration an alternate, less straight-forward approach that could handle thousands of compartments, provided no single user or file deals with too many at once.

As part of the task of implementing and enforcing security labelling of files we are also extending the concepts to objects other than files. Tape and disk volumes will be able to bear over-all security labels. A terminal and communication line will be able to be marked as to the highest security level and set of compartments accessible from it. Messages over communications media will be marked with the appropriate security labels, either by treating an entire session as being at a given level or by tagging each message, whichever is appropriate to the protocol involved.

As mentioned earlier, the TIP environment will also be modified to enforce access controls based on security labels, in addition to the current mechanisms.

At this point it is appropriate to mention that all our enhancements -- including those to be discussed below as part of the path to B2 and beyond -- are being done in a way that is upward compatible with existing user applications. This is not easy and has on occasion forced us to make uncomfortable design and implementation choices. In particular, under the new hardware architecture described below it will be necessary for quite some time to run a machine that has code operating in both the old and the new architectures.

OTHER SHORT-TERM ENHANCEMENTS

Although all are not strictly necessary to meet the letter of the Orange Book requirements, over the two-years or so it will take to do the primary level-B1 enhancements we will also be making a number of other changes and improvements in the security features of the system.

One set of enhancements and changes are associated with password management. The password file will be encrypted as a further protection against unauthorized access to it. We recognize in addition the need to provide further tools that encourage proper password use and discourage improper use. Tentatively it looks like we will support expiration times on passwords (to force users to change them) and system-generated passwords (to discourage users from picking easily-guessable passwords.) Unfortunately this is an area that seems to be frought with emotion: we have customers with very firmly-held views on the right way to manage passwords, and, as ought to be expected, a number of those views are incompatible.

One requirement (initially at the B1 level, but now at the C2 level) of the evaluation criteria (and of common sense) is that residue in storage media be erased before the space it occupies is allocated for a different purpose. The most efficient way to do this would be to ensure that a block of storage must by written by the user before it is read; an alternative is to scrub storage when it is de-allocated. A third alternative would be to allocate storage in pools, each pool of a single security level and compartment set. We haven't decided which approach to use for each of the various kinds of storage media concerned, but in each case not yet covered at least one acceptable approach has been identified.

Finally, to make the list complete, we are integrating the various parts of system administration, including security administration, into a common, system-wide interface.

TOWARDS B2 AND BEYOND -- NEW HARDWARE

Those of you who have studied the Orange Book know that the main feature that distinguishes the B2 and higher levels from B1 and the lower levels is the need to remove as much non-security relevant code and data from the Trusted Computing Base as possible, and, structuring what is left into independent modules. The strong implication of these requirements -- and even more so at the B3 level -- is that the separate modules of the TCB be isolated from each other by hardware

mechanisms, not just software engineering.

To meet this requirement -- although at the time it was only intuitively felt to be one, the Criteria not yet firm -- as well as other integrity and reliability needs, a number of significant changes were made to the hardware architecture of the 1100 series of computers; the first model of the series incorporating these changes was the 1100/90, which was first shipped to a customer last April.

Without going into a lot of detail, which you can read in the hardware manuals, I will outline the main changes relevant to this discussion. There are many other changes that make all this work efficiently, such as instructions to rapidly change addressing and protection context.

Base Registers

The 1100 series is a base-register machine, although those of you who have only a casual assembly-language reading knowledge of it may not be aware of that. The address space visible to a program is defined by four "bank" registers. A storage-referencing instruction, possibly after indexing, results in a "relative" address. Each bank register describes a region of the entire possible relative address space and a translation of that space into real, physical memory space. The relative address generated by an instruction is translated into a physical address by finding the bank register that includes that relative address; there are some subtleties to the algorithm that are exploited and some that are possible points of confusion, but the details are not important here.

What is important is that in the new architecture, called "extended mode", the user's addressing environment is now described by 16 base registers. (There is also a separate set of 16 base registers for use by the kernel of the operating system.) Storage referencing instructions operating in extended mode now contain a four-bit field explicitly selecting which base register to use. The physical address is formed by adding the relative address to a base value contained in the base register (after a limits check.)

Bank Descriptor Tables

In both the old architecture (called basic mode) and the new, the contents of base registers are protected. The operating system provides the user with a set of "bank descriptor tables" that describe all the possible banks it is legal for him to access. (Obviously, only the operating system can change the contents of a bank descriptor table.) The user has a set of instructions that load base registers from bank descriptor tables. One can think of the base registers as defining the user's current window into storage; the bank descriptor tables define everything he could potentially be allowed to see at any moment, and the load-bank instructions allow him to move his window around the visible portion, but only the visible portion.

In extended mode there are four bank descriptor tables that a user can load base registers from. (Incidentally, the four tables are in fact defined by four of the kernel's 16 registers.) Kernel software will structure the use of these to shape the entire virtual address space into a four-level tree. The top level contains banks which any user in the system can potentially see; the second level contains banks private to a single application, where an airline reservation system is an example of an application; the third level contains the banks for a program; and the fourth is private to a single process. The intention is that if a bank is to ever be shared by more than one part of the system, there will only be one copy of its descriptor, and that of course must reside high enough up in the tree for all potential users to be able to reference it.

Domains

The astute reader at this point will ask, "what about banks, such as those in the operating system, that must be accessible to all processes, but only when those processes are running in the operating system?" Associated with each bank (contained in its bank descriptor) is an access-control lock and two access permission fields, called general access permission and special access permission. Part of process state is an access-control key. Each access permission field encodes whether no access, read access, write access, or enter access is allowed. (Enter access is described below.) If the access-control key matches the lock, the access granted by the special access permission field is allowed, otherwise the general field applies. Thus the access to any bank can be differentiated into two kinds -- that allowed anyone who can reference the bank (which might be no access at all), and that allowed only when the process is running with the proper access key. The access lock and key are composed of a two-bit ring field and a seven-bit domain field. The two match if either the ring of the key is strictly less than the ring of the lock or if the two domain fields are equal. (If the domain field of the key is all zero, all forms of access regardless of the settings of the permission fields are granted.)

Gates

Obviously, control over when and how the access key can be changed is important. The key can only be changed by a Call or Jump instruction whose address is not an instruction itself, but rather a special construct called a Gate. The hardware knows the target of a Call is a Gate because contained in a bank descriptor is a field to indicate what type of bank is being described; for our purpose here, the main distinction is between gate banks and all other kinds. A Gate then is the means for making a controlled transition between domains of different privilege; one cannot enter a different domain without going through a gate, and one can only go through the gates for which one currently has Enter access permission. (Note that a Gate contains in addition to the access key to be used the instruction location to start at as well as bits controlling other parts of processor state.)

The hardware also supports a Return Control Stack, protected from any direct access by ordinary programs, on which is saved the privilege state, instruction location, and other information needed to return to the proper environment after a Call.

TOWARDS B2 AND BEYOND -- NEW SOFTWARE

To reach B2 and higher the main concept that we will be implementing beyond those needed for B1 is that of the software subsystem. Software subsystems are the means for structuring software into separate domains. A software subsystem is very much like a user: it has an identity, can be the owner and sole accessor of files and banks, and has security properties. It has its own dynamic linking environment and its own set of routines to respond to error conditions and software interrupts. When a process running in one subsystem attempts to link to another, the appropriate security rules are checked to see if the link is permitted; if so, and only if so, a gate only accessible to the calling subsystem is built that permits it to call into the second subsystem. Each subsystem is assigned a domain number, which is used to keep separate the banks belonging to the different subsystems.

The proper coupling between the software concept and the hardware mechanisms is being done very carefully. Just as an example, one would think that 7 bits of domain number would severely restrict the number of domains possible. In practice this is not true because it must also be remembered that in order for a bank to be accessed its descriptor must be in the part of the address tree that is addressable; in effect, each domain number is qualified by the node in the address tree it belongs to. There can thus be some fraction of the possible domain numbers that are shared across all parts of the system, a different fraction by each application, another by each program, and yet another by each process.

Ordinary users can write and use software subsystems, but those subsystems cannot have any special privileges -- i.e., they can access no more than their creator. Security administrators, however, can designate particular subsystems as being trusted, with the specific kind and degree of trust carefully controllable, ranging from merely permitting the subsystem to downgrade information to allowing it to execute with privileged hardware instructions.

With these software constructs we will be able to gradually restructure the system. More importantly, perhaps, users will be able to easily create their own protected subsystems and write multi-level applications. In both cases, whether or not the facilities are needed for and used for enforcing a security policy, they will certainly improve the integrity of the system and make software maintenance easier. Some parts of the software subsystem concepts will in fact be available long before the major B2-level restructuring.

CLOSING COMMENTS

There are two obvious omissions in the above: there is no discussion of security in our data management system, and there is no clear indication of when the B2 or higher level systems will be ready. Those omissions are deliberate.

We have some ideas on how to do data management security, but so far it is clear that our customers don't really know what they want or need above and beyond what is currently available. The National Academy of Sciences Summer Study on Secure Data Management Systems has been some help, but there is a lot of conceptual work yet to be done. As an interim step, we are considering having a single 1100 system support several data bases, each managed by one of our data management systems (we have both a relational and a CODASYL data management system), with each data base at its own security level separated from all the rest by the application-level nodes of the address tree.

We are getting a feel for what it is going to take to restructure the system to meet level B2 and above, but are not yet completely confident about it, nor about what priority we should place on it.

We are committed to continuously enhancing the trustworthiness of our systems. This report has covered the first few steps, in some detail. As you can see, we still have some choices to make and some priorities to set. We particularly welcome and solicit comments and advice in these areas and thank you for the advice we have already received, whether you think we have heeded it or not.

LIFE CYCLE ASSURANCE FOR TRUSTED COMPUTER SYSTEMS: A CONFIGURATION MANAGEMENT STRATEGY FOR MULTICS

Maria M. Pozzo

Honeywell Information Systems, Inc.

1 ABSTRACT

The goal of this paper is to discuss the control objective outlined in the Department of Defense Trusted Computer System Evaluation Criteria for the area of Life-Cycle Assurance in particular, Configuration Management. In addition, this paper will describe the Configuration Management Strategy for the development and maintenance of the Multics Operating System, and how it evolves over the life of the system, as the need for assurance of security mechanisms (documentation, source code, test procedures, etc.) becomes more prevalent.

2 INTRODUCTION

According to the Department of Defense Trusted Computer System Evaluation Criteria(1), six fundamental requirements are specified in the definition of a secure computer system. Of the six, four deal with the mechanisms needed to control access to information, while the remaining two discuss a means for obtaining assurances that these control mechanisms are provided and remain consistent throughout the life of the trusted computer system.

A control objective is a statement of intent with respect to control over some aspect of an organization's resources, or processes, or both [DOD 83], [FIPSP 80]. The Criteria outlines control objectives in three basic areas: security policy, accountability, and assurance. The intent of these control objectives is to provide a set of guidelines for developing a strategy for a specific system in order to meet the requirements of the Criteria.

Section 3 provides an overview of computer security and discusses the need for and definition of control objectives. Section 4 defines the control objective for Configuration Management, and discusses the various areas which must be covered in any such plan. Section 5 presents the current Configuration Management Plan for Multics and describes statistics that are gathered throughout this process. The gathering of statistics can help to identify success or failure of specific aspects of the plan so that they can be improved. It is in

---

(1) Order number CSC-STD-001-83, referred to as the "Orange Book" or "Criteria".

this way that the Multics Configuration Management Strategy evolves, as does the system itself, in order to continue to meet the requirements of both the Criteria and the expanding user community. Section 6 discusses the on-going evolution process for Multics Configuration Managment.

3 COMPUTER SECURITY

3.1 Background

Because the availability of computers has greatly increased in both the public and private sector, computer security is needed to protect against events that have adverse effects during computer processing. Towards this effort, the DoD Computer Security Center was founded to provide guidance and encouragement to vendors who develop and maintain commercially available secure computer systems. According to the Criteria, a secure system is one that provides specfic mechanisms, both automated and manual(1), which control access to information such that only authorized individuals, or processes running on behalf of individuals, have access to read, write, create or delete such information. In a broader sense, computer security also includes assurances that the system functions properly and is maintained in such a way as to prevent harmful side-effects [FIPSP 80].

3.2 Security Objectives

During the initial planning stages for any system, it is necessary to first define the objectives of the system. For a secure computer system, the Criteria defines three such objectives: security policy, accountability and assurance. Each objective defines a set of one or more controls which must be present in a system in order to achieve a particular security objective. The first area, security policy control objective, defines four such controls. To achieve this objective, a system must provide controls with respect to the system's security policy in general, the system's mandatory access controls, the discretionary access controls, and the system's policy on marking information. Each of these controls is identified as a control objective. The accountability and assurance objectives each have one such control to achieve the respective objective. Therefore, acording to the Criteria, there are six control objectives: Security Policy, Mandatory Security, Discretionary Security, Marking, Accountability, and Assurance.

Each control objective defines a set of requirements which provide a framework for meeting the security control objective. For example, the Criteria defines an accountability objective for secure systems.

---

(1) Manual mechanisms are those procedures recommended by system documentation in order to maintain a trusted environment in which the system must be operated in order to provide a specific level of security.

There are several requirements, 1) individual user identification, 2) authentication of the user identification, and 3) dependable audit capabilities. The accountability control objective as defined in Section 5.3.2 of the Criteria provides a framework that is flexible enough to encourage a variety of mechanisms that will meet this security control objective.

Section 4 will provide an in-depth discussion of the Assurance Control Objective as defined by the Criteria.

4 LIFE CYCLE ASSURANCE

The control objective as stated by the Criteria for assurance is stated below. The Criteria specifies two requirements for the assurance control objective: operational and life-cycle assurance. This paper deals only with life-cycle assurance.

> ASSURANCE CONTROL OBJECTIVE
>
> Systems that are used to process or handle classified or other sensitive information must be designed to guarantee correct and accurate interpretation of the security policy. Assurance must be provided that correct implementation and operation of the policy exists throughout the system's life-cycle.

The life-cycle of a computer system has three phases: initiation, development, and operation (that phase at which the system is accepted and used as intended). After some period of operation, the system will be expanded or revised and the life-cycle begins again [BRAND 82]. Security must be considered at each phase of the computer system's life-cycle [FIPSP 80].

4.1 Configuration Management Strategy

Configuration Management is a policy statement which controls an organization's resources and procedures during a computer system's life-cycle. In order to meet the requirements of the Criteria, a Configuration Management Plan must cover all aspects of the computer system's life-cycle from initiation to operation. Specifically, a Configuration Management Plan applies to expansions or revisions of a computer system. The following section describes the areas that must be covered under such a plan and provides possible suggestions for developing a Configuration Management Strategy.

4.1.1 INITIATION PHASE

During the initiation stage, a variety of alternatives must be considered with respect to the intended revision or expansion. Feasibility studies and cost-benefit analysis should be conducted. The impact on the computer system's security mechansims must be evaluated during this phase [FIPSP 80]. For example, suppose a revision is needed due to a system bug, customer request, or general enhancement. The initiation phase should first perform a feasibility

study to determine a set of alternatives for making the change (one of which might be not to make the change). Each alternative includes its cost and hardware, software and security implications. It is then possible to perform a cost-benefit analysis. The set of alternatives will be larger for a new system than for one already in existance, which will be restricted by its current implementation. See [FIPSP 80] for more details on the initiation phase.

4.1.2 DEVELOPMENT PHASE

Once a commitment has been made to revise a system, the development phase can begin. This phase has the following stages: definition of the high-level description, detailed design, implementation and testing. A Configuration Management Plan defines procedures for all stages of this phase from high-level description to testing.

4.1.2.1 High-Level Description

As a first step, the developer provides a high-level description of the revision, documents it and submits it to management and peers for review. This description includes any relevant security issues and their implications as well as areas of the system that could be affected by the change. Since this is a high-level description, details of interfaces are not necessary. This stage should be completed prior to any prototyping.

4.1.2.2 Detailed Design

Subsequent to the high-level description, it may be necessary to perform some initial prototyping. A prototype is a skeletal implementation which aids the developer in building the design and may greatly impact it. The prototype is not part of the implementation and is usually discarded once the design is completed. With the aid of prototype results, the high-level description, and a thorough knowledge of the area of the system being revised, the developer proceeds with the detailed design. A detailed design document describes the current state of the area to be revised, the reason it is being changed (such as known problems), and a detailed description of the change. All interfaces should be specified, particularly those interfaces external to the area being changed. If the change is for a part of the system which involves the security mechanisms or may have some impact on overall system security, these issues should be described in detail. Testing procedures should also be included. Lastly, a plan for completing the change should be specified. This plan should include the effort required for each phase of the revision.

4.1.2.3 Design Review

Once the design document has been completed, a design review is conducted. The design should be reviewed by several different groups of individuals. One or more persons knowledgeable in security issues

should conduct a review geared towards the security implications. A peer review should consider the technical aspects of the design. Finally, a management review should determine if the design is acceptable. This stage is complete when the design document has been updated to reflect the results of the design review.

4.1.2.4 Implementation and Testing

Implementation should adhere to the organization's programming standards. Programming standards specify the language(s) which can be used, correct programming formats and conventions, and development tools which are recommened. Programmers should be sure to implement only that which is specified in the design document. If this is not the case, the design document must be updated and reviewed as stated in the previous paragraph. Finally, complex or overly sophisticated code may cause more problems than are justified by the efficiency they are intended to achieve. In most cases, simple, straightforward code provides the best implementation.

Testing should proceed as specified in the design document. There are many ways to accomplish testing. One example is to test at the interface level. When a system is tested through its interface, its behavior is tested to insure that it performs those functions and only those functions that it is intended to perform. Testing can be accomplished manually or automatically. However testing is conducted, a test plan should be provided which states the tests to be made and their expected results.

There must be a set of procedures for exposure. One thing that must be considered is where in the file system the development work is to be accomplished. Depending on the nature of the work, it may be necessary to provide security mechanisms so the code under development cannot be tampered with. If modifications must be exposed, there must be a standard place in the system where such experimental code can reside. The length of time for exposing such code must also be specified.

4.1.2.5 Audit

When testing and implementation have been completed, the code should be audited. For security-related areas of the system, it may be necessary to have the code audited by a security expert initially, followed by a technical audit. The auditor should consider programming standards to insure that the developer has followed the organization's conventions and recommended procedures during implementation. The auditor should understand what has been changed, compare it to the design document, run the tests, etc. Part of the Programming Standards document should state the procedure to follow during audit.

4.1.2.6 Completion of the Development Phase

Before the development phase can be completed, documentation must be provided or updated as appropriate. The developer must insure that any design documentation clearly reflects the implementation. All user documentation must also be written or updated.

Lastly, a set of procedures must be provided for installation of the new code as well as release to customers. Installation may be different depending on the reason for making the change. It is important, however, to keep records of installations so future modifications to the same area of code can be performed correctly.

4.1.3 OPERATION PHASE

During normal operation, there are many reasons for revising or expanding a computer system. The Configuration Management Plan must provide procedures for the different types of changes. For example, if the modification is emergency in nature, there may not be time to follow all the above-outlined steps to the same level of detail as for an enhancement. The strategy must clearly identify the procedures to follow for all such cases.

5 MULTICS CONFIGURATION MANAGEMENT

During the early days of Multics, various parts of the Configuration Management procedures were documented in separate Multics Administrative Bulletins (MABs)(1). Over the years, the MABs have been updated, revised and new ones written. In more recent times, these MABs have been consolidated into one definitive document and published internally as a Multics Technical Bulletin (MTB)(2). Currently, this MTB is being expanded and revised for eventual release as an official Multics Administrative Bulletin on Configuration Management Procedures and Policies.

5.1 Initiation Phase

Individual members of the Multics community (contributors) start changes for one of several different reasons: 1) they have an idea for an improvement to the system, 2) they respond to a requirement formally presented through management channels, 3) they respond to a reported bug or suggestion.

Upon receipt of a change request, management determines its feasibility and assigns the change to a particular developer. The

---

(1) Multics Administrative Bulletins contain Honeywell proprietary information and are not available to the public.

(2) Multics Technical Bulletins contain Honeywell proprietary information and are not available to the public.

developer's responsibility is to perform any research necessary to produce the appropriate design documentation.

5.2 Development Phase

5.2.1 DESIGN DOCUMENTATION

To propose a change a developer produces an MTB, which is a detailed description of the proposed design, or a Multics Change Request (MCR), which formally requests technical approval of a change. An MCR must always be written while an MTB need only be written under certain conditions: 1) the modification will cause a change to the central design philosophy of the system, data structures, or security policy of the system or one of its subsystems, 2) the change will add a new subsystem to the system(1), 3) the documentation required will be more than several pages, 4) the change involves design issues where the developer is unsure of the correct solution, or where others are likely to disagree.

In either case, the design document must describe any new or changed user interfaces to the system; any new or changed internal interfaces that are usable outside of the subsystem; any significant new or changed data structures, design philosophies, in particular security-related issues. The level of detail supplied by an MTB is often dependent on how crucial the changed area is to the system, for example, MTBs for areas of the system where a problem exists often provide a description of the current state and reasons for the intended change.

As described above, some implementation, in the form of a prototype, is often done prior to completion of the design.

5.2.2 DESIGN REVIEW

A draft of the MTB is made available to peers for technical review, and to an individual knowledgeable in security for security issues. When all resulting questions, complaints, and suggestions have been resolved by the developer, the MTB is updated to reflect the results of the design review and published internally.

The next step is to fill out the standard MCR form and submit it for approval. In the event that the change did not meet the criteria outlined above for writing an MTB, this would be the first document specifying the design for the change. Changes that only require an MCR are usually small and do not effect a large portion of the system. The MCR form requires the signatures of two individuals, the developer's manager and a sponsor. The sponsor checks that the MCR is

---

(1) A subsystem is a significant body of code that provides a set of related functions, and has a clear modularization that separates it from the rest of the system.

complete and takes responsibility for its technical plausibility. If the change has security implications, the sponser is usually an individual with knowledge of security issues.

When the MCR has been completed and approved according to the above steps, the MCR is submitted to the General MCR Board (GMCRB) on which all developers in the Multics organization can participate. The GMCRB discusses, reviews and votes on the MCR. If the vote indicates a lack of consensus, the MCR is submitted to the Executive MCR Board (EMCRB), which is a much smaller group, appointed by management, and has the final authority to resolve technical issues.

5.2.3 IMPLEMENTATION

Once the MCR Board approves the MCR, the developer can begin implementation being careful to implement only that which is specified by the design documentation. Because of the nature of the Multics Trusted Computing Base (TCB), that portion of the system which implements and protects the security-relevant code, it is imperative that developers follow proper programming practices. The Multics Programming Standards System Designers' Notebook (SDN) (Order Number AN-82) provides guidelines for writing efficient, readable Multics PL/I(1) programs. Currently, this document is undergoing revision, since guidelines for writing ALM programs, and programs that run in special environments, are not included in the SDN.

The implementation phase includes testing and limited exposure. Generally, developers can expose their code on the CISL Multics System, or in experimental libraries on MIT (Cambridge, MA), CISL (Cambridge, MA), ACTC (Calgary, AB), or System-M (Phoenix, AZ).

5.2.4 AUDIT

The audit verifies that the implementation meets its requirements as specified in the design document. An auditor must first review the MTB and/or MCR provided by the developer. The auditor then insures that the code conforms to the design and proper programming practices, has no apparent bugs and has been adequately tested. If the change has security implications, the code may also be audited by a security expert. Prior to installation, the developer must fix any problems found by the auditor.

5.2.5 INSTALLATION

Multics Administrative Bulletins are available which provide a set of standards and procedures for preparing installations to be made by the Installation Group in Phoenix. The developer must first fill out the proper installation forms which contain the changed or new modules,

---

(1) Most of Multics is written in the high-level language PL/I while the remainder is written in Multics Assembler (ALM).

the approved MTBs and/or MCRs, and the signatures of the developer, auditor, and the developer's manager. All new or changed documentation must be sent to the Documentation Group and, where appropriate, the installation forms must include a signature from the Documentation Unit manager.

The installers examine the forms for completeness, verify that all affected modules have been included and recompile the source code provided by the developer. Tests provided by the developer as well as any other appropriate tests are run prior to installation into the System-M libraries.

5.3 Operation Phase

A new release of Multics occurs approximately once every 12 months. During the release cycle, a number of problems can arise that warrant changes to the system prior to the next formal release.

5.3.1 EMERGENCY PROCEDURES

Emergency Change Requests (MECRs) are used to submit emergency changes to exposure sites while Critical Fixes are used to make emergency changes available to the field. MABs are available internally which specify the procedures to follow for emergency changes.

Fixes of the above nature are generally done without any design phase since rapid response is often imperative, however, an audit is performed prior to installation on System-M in Phoenix. Once the change has been installed, the developer must provide an MCR, conduct a design review and resubmit the change as a normal installation.

5.3.2 STATISTICS

Gathering statistics during the life-cycle of a system can help to identify areas of the Configuration Management where there are weaknesses. One such measure is to examine the number of MTBs and MCRs that are rejected or submitted to the EMCRB due to controversial issues. If this number is large, it can be an indication that a high-level description was not provided or thoroughly reviewed prior to any prototyping and formal design work. Another measure is the number of times emergency procedures must be invoked. If emergency procedures are used often, testing procedures may be inadequate, and more stringent policies for testing changes must be developed.

If the available documentation is largely out of date, this is a sure indicator that the procedures for supplying sufficient, up-to-date documentation are inadequate.

If auditors are continually rejecting code there may not be a consistent understanding among developers of proper programming practices. Overall, managers must keep a close watch for indications that there are problems in any of these and other areas.

6 GROWTH OF THE MULTICS CONFIGURATION MANAGEMENT PLAN

Multics is currently under evaluation by the DoDCSC; as of this writing, the evaluation is not yet complete. In order to provide a system according to the Criteria, and to meet the needs of the growing Multics community, several areas of the Multics Configuration Management Plan are under consideration for revision and addition.

For Multics, adequate documentation for code within the TCB is an important addition. For future design documentation, the format of Program Logic Manuals (PLMs) has been modified contain a chapter describing the overall subsystem, how it interfaces both internally and externally; and a chapter that discusses the security policies and implications of the subsystem.

The Criteria requirements specify that testing must be performed at the interface to the TCB and provide adequate assurance that the system performs those functions that it is intended to perform. Multics Configuration Managment Policies are being expanded to include a set of procedures for providing tests which satisfy the requirements of the Criteria more adequately than current testing procedures.

Currently, a proposal is under consideration that would require developers to submit a high-level description of the proposed change prior to generating a detailed design. A review of the high-level description would provide developers with input as to the technical feasiblity and possible acceptance of the future MTB and/or MCR before a substantial amount of work has been done.

7 CONCLUSION

This paper has attempted to show areas that must be considered when developing a Configuration Management Plan. Such a plan for Multics has been presented as an example. Any such plan must be open to modification as the needs of the organization and its user community change. For Multics, the evaluation process conducted by the DoDCSC, has helped to point out areas for improvement. "Much of what needs to be done to improve security is not clearly separable from what is needed to improve usefulness, reliability, effectiveness, and efficiency of the computer...[system]". [FIPSP 80]

REFERENCES

[BRAND 82] Brand, S. L. "An Approach to Identification and Audit of Vulnerabilities and Control in Application Systems," in Audit and Evaluation of Computer Security II: System Vulnerabilities and Controls, Z. Ruthberg, ed., NBS Special Publi. #500-57, MD78733, April 1980.

[DOD 83] DoDCSC "Trusted Computer System Evaluation Criteria", CSC-STD-001-83, August 1983.

[FIPSP 80] Federal Information Processing Standards Publication, (FIPS PUB) 73, Guidelines for Security of Computer Applications, 30 June 1980.

GOULD SOFTWARE DIVISION'S SECURITY PROGRAM

Gary Grossman
Vice President, Research and Development
Gould Software Division
Urbana, Illinois

INTRODUCTION

Gould Software Division (GSWD) began as Digital Technology Incorporated (DTI) in 1977. It became Compion Corporation in 1982, and was acquired by Gould, Inc. in 1983. As DTI and Compion, GSWD primarily produced network front ends for the U.S. Defense Communications Agency. This work formed the basis for the Secure HUB™ Executive, a formally verified multi-level secure operating system, and the Communications Operating System Network Front End (COS/NFE), a formally verified multi-level secure network front end. The experience gained through participation in verifying the HUB™ and the COS/NFE was applied in producing the VERUS™ formal verification system.

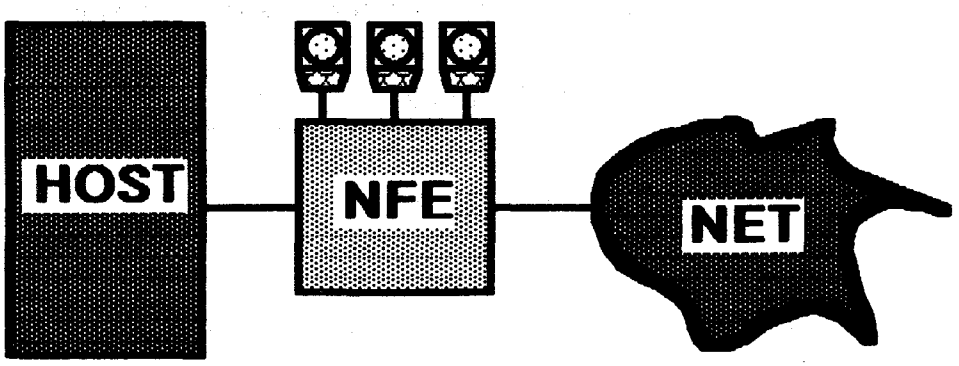
GSWD has a continued commitment to security technology and security products. VERUS™ is a fully supported product with an aggressive enhancement program. The Protector™, the first in a series of security products aimed at the commercial computer and communication security market, will be released late this year. Enhancements to the Secure HUB™ Executive are being investigated. GSWD has expanded the security properties of UNIX™ for use in the DoD and in the commercial market. A further security-enhanced version of UNIX™ is scheduled for release in 1985. And GSWD is developing a formally verified multi-level secure version of UNIX™.

HISTORY

Non-Secure Network Front Ends

As DTI, GSWD's business was primarily producing network front ends for the Defense Communications Agency. A network front end (NFE) is a computer system interposed between a mainframe computer and a resource sharing network (see Figure 1). The NFE may also support terminals that can connect to both the mainframe and the network. DTI personnel were involved in three non-secure NFE projects intended to connect World Wide Military Command and Control System (WWMCCS) Honeywell H6000 hosts to various networks:

1. The Experimental Network Front End[1] was developed at the University of Illinois to connect an H6000 with the ARPANET. Its hardware base was a Digital Equipment Corporation PDP-11/70; its software base was the UNIX™ time-sharing operating



Network Front End

Figure 1.

system[2], with software added to implement the ARPANET protocols and to augment the UNIX™ interprocess communication facilities.

2. The Interim Network Front End[3] (INFE) was a prototype developed at DTI to connect an H6000 with the AUTODIN II network. It was also based on the PDP−11 and on UNIX™, with the DoD standard TCP/IP protocols and with a new interprocess communication mechanism called Attach I/O[4].

3. The WWMCCS Network Front End[5] (WNFE), also produced at DTI, was designed to be fielded as an NFE that connects an H6000 with the WWMCCS Intercomputer Network. It used the same hardware and software bases as the INFE.

GSWD's network front end projects are summarized in Figure 2.

Security Projects

As DTI and Compion, GSWD completed two formally verified secure software projects: the Secure HUB™ Executive and the COS/NFE; Compion also developed the VERUS™ verification system.

Secure HUB™ Executive. The Secure HUB™ Executive[6] was developed by DTI in 1980 as a formally verified secure operating system oriented toward supporting communications and other real−time applications. The HUB™ was designed to support DoD multi−level security while providing better performance than had been available through performance−enhanced versions of UNIX™. HUB™ interprocess communication was twice as fast as that of UNIX™; overall system performance was 20% better with the HUB™. The HUB™ was designed to be portable; it was first implemented on the INFE hardware base and also on a Motorola M68000 microprocessor.

Project	Org.	O/S	Network	Goal
ENFE	CAC	UNIX	ARPANET	Prove concept
INFE	DTI	UNIX	AUTODIN II	Interim
WNFE	DTI	UNIX	WIN	Deployable
COS/NFE	DTI	HUB	AUTODIN II	Secure

GSWD Network Front End Projects

Figure 2.

COS/NFE. The COS/NFE[7] was completed by Compion in 1983. It was designed to provide the same functions as the INFE, but with formally verified multi-level security based on the Secure HUB™ Executive. These goals were met, with increased performance, on the INFE hardware base.

VERUS™. The VERUS™ formal verification system[8] was developed by Compion in 1982. It then consisted of a parser for a specification language that was based on the first order predicate calculus with types, and a theorem prover. VERUS™ was used to respecify and reprove the Secure HUB™ Executive and the COS/NFE.

STATUS

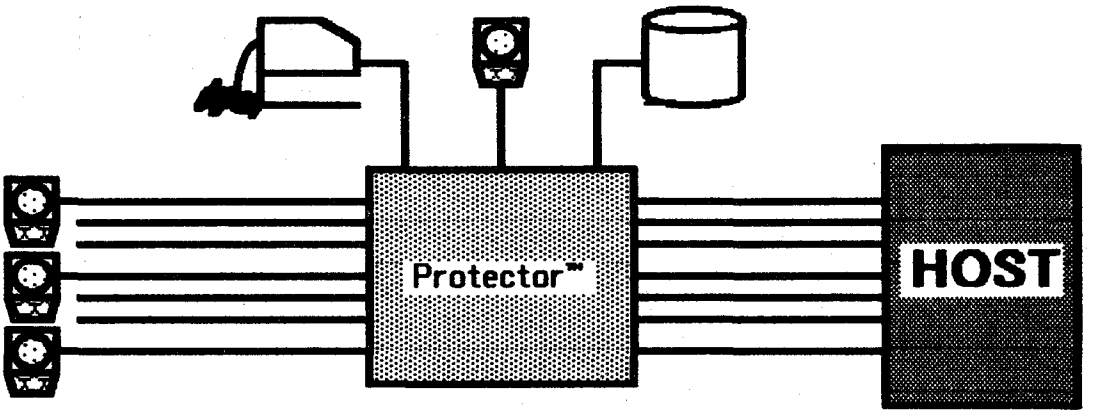
GSWD has built on the technology it has developed to produce products that are now on the market or are soon to be released. These include the Secure HUB™ Executive, the Protector™, VERUS™, and security-enhanced UNIX™.

Secure HUB™ Executive

The HUB™ is a product that is available under license as a building block for secure systems. It has been implemented on a variety of supermini-, mini-, and micro-computers.

The Protector™

The Protector™ is a communications and security product that is based on the technology developed for the COS/NFE. It provides a tamper–proof audit of computer use to deter abuse through threat of exposure.



The Protector™ Concept

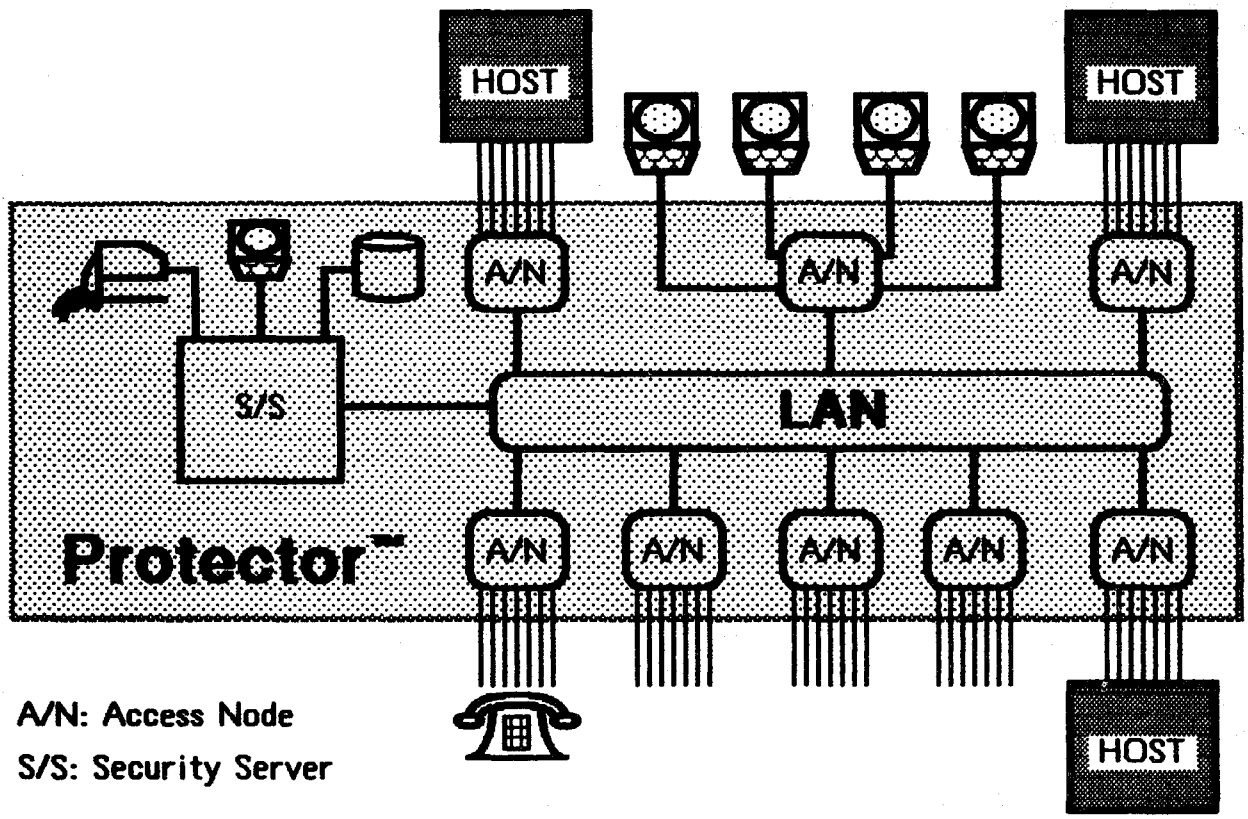
Figure 3.

Conceptually, the Protector™ is interposed between a host and its terminals, much like a network front end (see Figure 3). The Protector™ provides a tamper–proof audit trail of user input based on positive user authentication. The audit trail for each user is captured according to data patterns entered by a site administrator or security officer. The audit trail is stored on disk, where it can be selectively examined on–line by the security officer. The security officer can also copy the audit trail, or selected portions of it, to a printing device. The Protector™ controls access to the host by individual user and by other criteria such as time of day and physical location of the user terminal.

In actual implementation, a Protector™ system consists of a set of access nodes and a security server, all interconnected by a local area network (LAN) (see Figure 4). The access nodes are connected to hosts and terminals via standard interfaces and protocols. They interconnect the terminals with the hosts via logical connections over the LAN. The security server makes all security–relevant decisions regarding user authentication and access control, and provides storage for access control information and the audit trail itself.

VERUS™

VERUS™ is a fully–supported product with complete documentation, maintenance, and a newsletter. Integers have been added to the specification language as a type, and theorems about them are handled by the prover. A specification checker has been added to ensure that all requisite theorems for a given specification are stated and proven.



The Protector™ Implementation

Figure 4.

UNIX™ Security Enhancements

Substantial enhancements have been made to the security support provided by Gould's UTX–32 operating system, which is based on Berkeley 4.2 BSD UNIX™ with additional features of AT&T's System V. These enhancements include

1. security labels on files, directories, directories, devices, users, and processes;

2. additional rules controlling access of processes to files;

3. expanded user authentication procedures; and

4. expanded accountability facilities.

FUTURE PLANS

GSWD has planned an aggressive program of enhancement of its current security-related products, as well as introduction of new products.

Secure HUB™ Executive

The HUB™ security policy and verification evidence will be re-examined in the light of the security evaluation criteria that have been developed since the HUB™ was completed.

Protector™

The Protector™ will be enhanced to prevent user input from entering the host if the data matches patterns specified by the security officer on a per-user basis. Traffic on the Protector™ LAN will be encrypted to further secure communication within the Protector™ system. Additional communications media such as long-haul networks will be supported. Provision will be made for securing individual remote terminals.

VERUS™

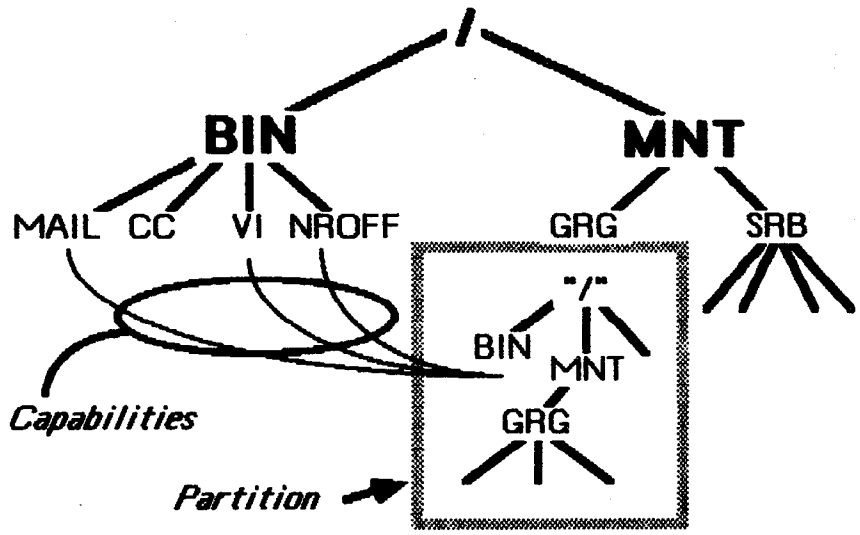
Expert system technology will be employed to enhance the ease of specification and proof, possibly extending as far as automatic proof construction. Extension of VERUS™ to source code proofs will be investigated.

UNIX™ Partitions and Capabilities

UNIX™ security will be further enhanced through two features, partitions and capabilities, now under development at GSWD (see Figure 5).

A partition encloses a portion of the UNIX™ hierarchical directory tree; it appears to a user as if the enclosed portion were the entire UNIX™ file system. The highest node in the hierarchy within the partition appears to be the root ("/") of the file system. It is not possible for the user to access anything outside the partition because all file pathnames are interpreted relative to the apparent root within the partition. This is true even for users with "superuser" privileges. The partition facility is currently used at GSWD to permit outside users to dial in to company UNIX™ systems that contain sensitive corporate data. Each outside user group is given its own partition; data and programs that are to be shared with these users must be copied into the appropriate group's partition by a privileged company user.

If every user of a system were given a unique partition, then users could not share data at all. In fact, each user's partition would have to contain a copy of all the command programs and system data files that the user was to access. This would consume a large amount of disk storage space to support even a moderate number of users.



UNIX Partitions and Capabilities

Figure 5.

The capability facility provides a solution to these problems by providing a controlled means of accessing directories and files outside a user's partition. A capability is an object in the directory tree within a partition that names a directory or file outside the partition. Capabilities would be placed in each user's partition that name only those files and directories that the user is to access. This would permit a privileged user such as a security officer to selectively grant individual user access to certain system files while denying access to other files. For example, a user could be given access only to programs that provide text editing and mail facilities, but no access to program creation facilities such as compilers or link editors.

A capability provides a separate access control specification that may be more restrictive than the access control that is directly associated with the directory or file itself. For example, a log file that is specified to be readable and writeable by anyone on the system might have to be accessed by a particular user via a capability that permits only writing. This would permit the user's activities to be logged without giving the user access to log information about other users.

Multi-level Secure "UNIX™"

GSWD is studying the problem of developing a UNIX™-based system that would be governed by the full Bell and LaPadula security model[9]. This system would be intended for evaluation by the DoD Computer Security Center[10] initially as a candidate for Class B2, and eventually as a Class A1 system.

Developing a system of this kind involves more than formal specification, careful implementation, and formal verification. UNIX™ contains many mechanisms that are antithetical to features of the full Bell and LaPadula model, particularly the "*-property".

Discussion of these issues is beyond the intended scope of this paper, but it appears that a fully multi-level secure system would retain many basic paradigms and features of UNIX™, but would differ significantly in visible mechanisms and in the details of use.

GSWD intends to aggressively seek solutions to these problems and to pursue a program leading to a multi-level secure UNIX™-like system.

REFERENCES

1. Holmgren, S.F., et. al., Experimental Network Front End Functional Description, 7502, Command and Control Technical Center, WWMCCS ADP Directorate, Defense Communications Agency, Washington, DC, January 1977.

2. Ritchie, D.M. and K. Thompson, "The UNIX Time-Sharing System", Communications of the ACM, Vol. 17, No. 7, July 1974, pp. 365-375.

3. Grossman, G.R., S.F. Holmgren, and R.H. Howe, INFE Functional Description Overview, Document 2, Digital Technology Incorporated, Champaign, IL, March 1978.

4. Attach I/O User Manual, 78019.C-INFE.12, Digital Technology Incorporated, Champaign, IL, October 1978.

5. Allen, E.R. and R.H. Howe, WNFE Functional Description, 81045.C-WNFE.19, Champaign, IL, April 1982.

6. Grossman, G.R., "A Practical Executive for Secure Communications", Proceedings of the 1982 Symposium on Security and Privacy, IEEE Cat. No. 82CH1753-3, Oakland, CA, April 1982, pp. 144-155.

7. Grossman, G.R. "COS/NFE - A Multi-Level Secure Network Front End", Proceedings of the Digital Equipment Users Society, Atlanta, GA, May 1982, pp. 1205-1221.

8. Marick, B., "The VERUS™ Design Verification System", Proceedings of the 1983 Symposium on Security and Privacy, IEEE Cat. No. 83CH1882-0, Oakland, CA, April 1983, pp. 150-157.

9. Bell, D.E., and L.J. LaPadula, Secure Computer Systems: Mathematical Foundations and Model, M74-244, The MITRE Corp., Bedford, MA, May 1973.

10. Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83, DoD Computer Security Center, Ft. Meade, MD, August 1983.

ELECTRONIC FUNDS TRANSFER SECURITY

Dick Bauder
U.S. Treasury

Securing Government Electronic Funds Transfers

Current payment systems
Current EFT security
Treasury Directive 81-80
Impact on current systems
EFT security implementation

Current Payment Systems

Standard payments

- SF 1166
- payment distribution - check/ACH

Treasury Financial Communications System (TFCS)

- host-to-host interface - FRBNY
- host/slave with federal agencies
- DLA host-to-host pilot

Current EFT Security

ID and password
ID restricted to individual terminal ID
ID restricted by functions
Automatic disconnect
Encryption

Treasury Directive 81-80

"... all EFT's must be properly authenticated."
- ANSI X9.9 - financial institution message authentication
- equivalent authentication techniques

Who does directive apply to?

"... All systems which originate, transmit, relay, receive or process, federal government EFT transactions..."

Equipment guidelines

"Equipment designed and used to perform the authentication function must comply with Federal Standard 1027..."

Effective Date

- all current systems by June 1, 1988
- all systems implemented after August 16, 1984 must comply immediately

Federal Payments Systems Impacted by the Policy

Treasury Financial Communications System (TFCS)
Automated Clearing House (ACH)
Direct Deposit (DD/EFT)
Checks
Electronic certification program
Army ATM pilot

Federal collections systems impacted by the policy

Treasury general account cash concentration system
Farmers home administration cash concentration system
Treasury tax and loan account system
Treasury lockbox system
Treasury financial communications system (TFCS)
Automated clearing house (ACH)
- preauthorized debits
- corporate to corporate
- home banking

Federal securities systems impacted by the policy

Commercial book entry systems at Federal Reserve Banks
Treasury direct access book entry systems

Other payments systems that may elect to adopt the policy

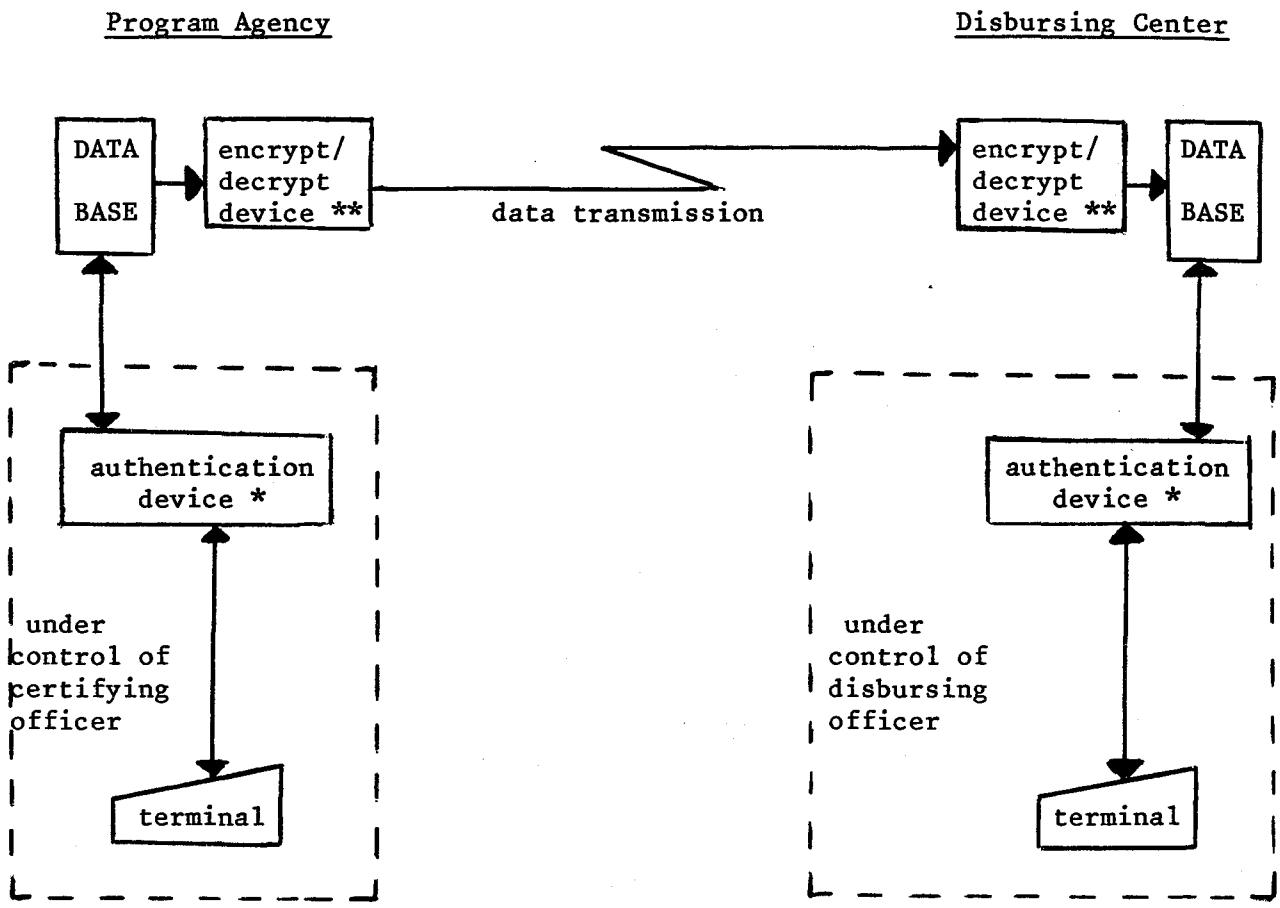
Federal Reserve Communications System after 1980 (FRCS-80)
Clearing House Interbank Payments System (CHIPS)
The Society for Worldwide Interbank Financial Telecommunications (SWIFT)
Bankwire
Bankers Automated Clearing Services (BACS)
Clearing House Automated Payments System (CHAPS)
Postal money transfer system (GIRO)

Impact on Current Systems

ACH
- electronic certification
- message authentication

TFCS
- initial implementation - point-to-point
- enhanced implementation - end-to-end, center

ELECTRONIC CERTIFICATION SYSTEM CONFIGURATION
USING MESSAGE AUTHENTICATION AND ENCRYPTION



* This component may or may not be an integral component of the Data Terminal

** Required if data is sensitive and must be protected from a monitoring threat

EFT Security Implementation

Equipment certification - Treasury/NBS/NSA
Certification guidelines
Implementation task force - Treasury/NBS/NSA/FRB/GAO
BGFO is executive agent
Implementation schedule

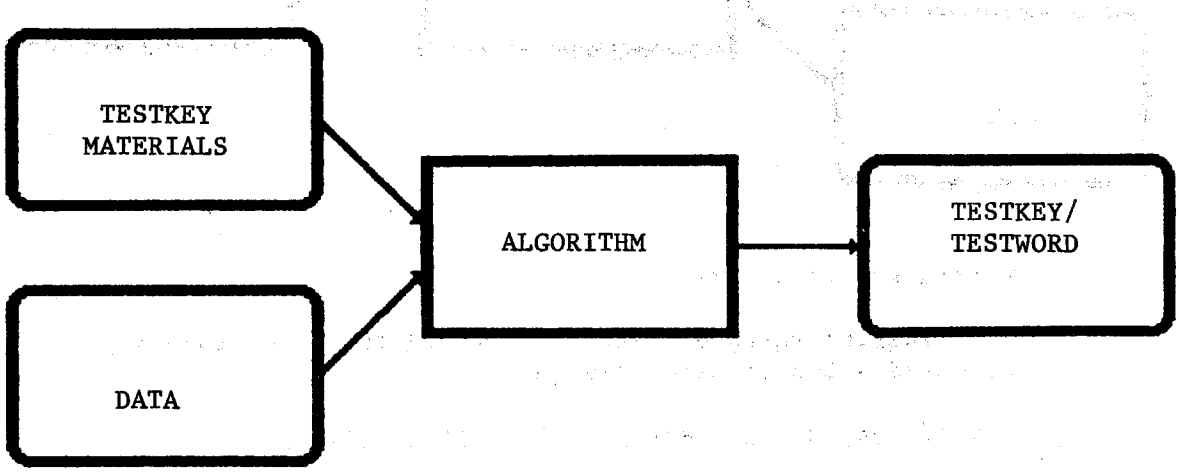
EFT SECURITY IMPLEMENTATION

Proj. No. and/or Description work to do; action to take	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1. Memorandum of understanding between Treasury/NSA/NBS	•															
2. Certification Guidelines	•	-	-	•												
3. Treasury Task Force	•	-	-	•												
4. Certification of Equipment				•	-	-	-	-	-	-	-	-	-	-	-	-
5. BGFO System Implementation																
6. o System Design o Systems Development		•	-	-	-	•										
					•	-	-	-	-	•						
7. o Install Initial System Pilot o Test and Evaluation										•	-	•				
												•	-	-		
8. o Production System Plan																
9.																
10.																
11.																
12.																
13.																
14.																

AUTHENTICATION of ELECTRONIC FUNDS TRANSFERS

Richard Y. Yen
The Chase Manhattan Bank, N.A.

TESTKEY SYSTEM



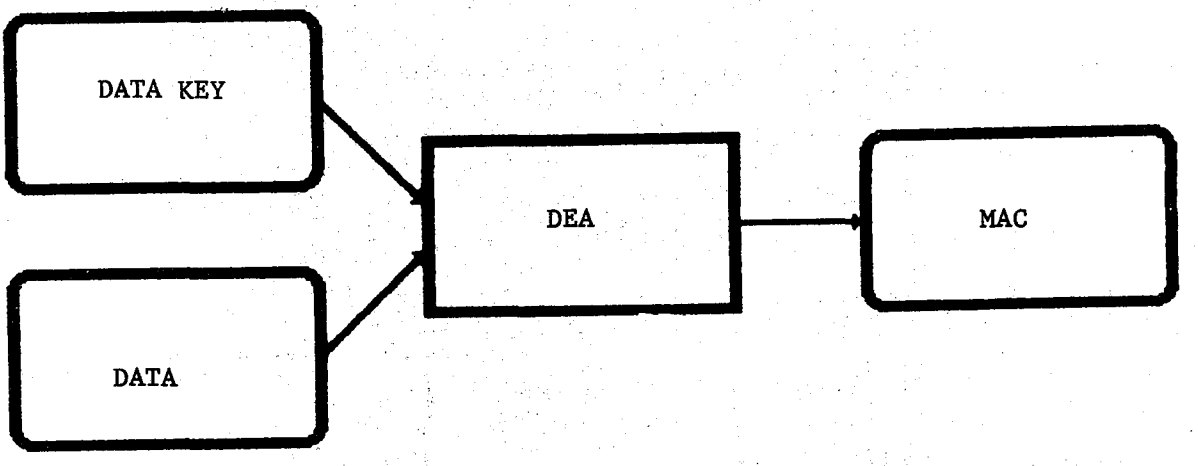
Present Testkey System Provides:

- o redundancy check on value(dollar amount)
- o information on message loss or duplicate message(if sequence number is used)
- o limited security

Operationally it is Difficult to Exchange Testkey Materials often to Ensure Security

What are the Alternatives?

MESSAGE AUTHENTICATION



- o A data key is selected
- o Text editing rules are applied for the entire message text or selected authentication elements
- o Edited data are processed by the DEA algorithm
- o The resultant MAC is added to the message before transmission
- o The recipient selects the same data key and repeats the extraction and computation processes, and accepts the message as authentic if the MAC is identical to that in the received message

THE ANSI FINANCIAL INSTITUTION MESSAGE AUTHENTICATION STANDARD (FIMA)

- o FIMA can protect either selected fields or entire messages
- o FIMA can be used for funds transfers or textual messages (such as syndicated loan agreements)
- o FIMA can be used with any data communication system

Within an institution
- branch-to-branch
- customer cash management services
- programs(including downline load and audit)

Inter-institution
- fedwire
- bankwire
- swift
- common carriers
- packet networks, etc.

- o FIMA provides very high security against message modification and message insertion for those message elements protected

- o FIMA uses the NBS Data Encryption Standard(DES) to compute a message authentication code(MAC). The MAC replaces the testkey function.

MAJOR CONTROLS FOR IMPLEMENTATION

Key Management

- o Security is provided by keeping key secret

- o Key management is the most difficult problem for users of authentication and encryption systems

- o This problem is especially difficult for geographically widespread computer communications networks

Physical Security of the Equipment

The equipment shall be protected against

- unauthorized access to keys
- unauthorized operation of the equipment
- unauthorized modification of the equipment

INFORMATION SYSTEMS SECURITY AT SECURITY PACIFIC NATIONAL BANK

Ed Zeitler
Security Pacific National Bank

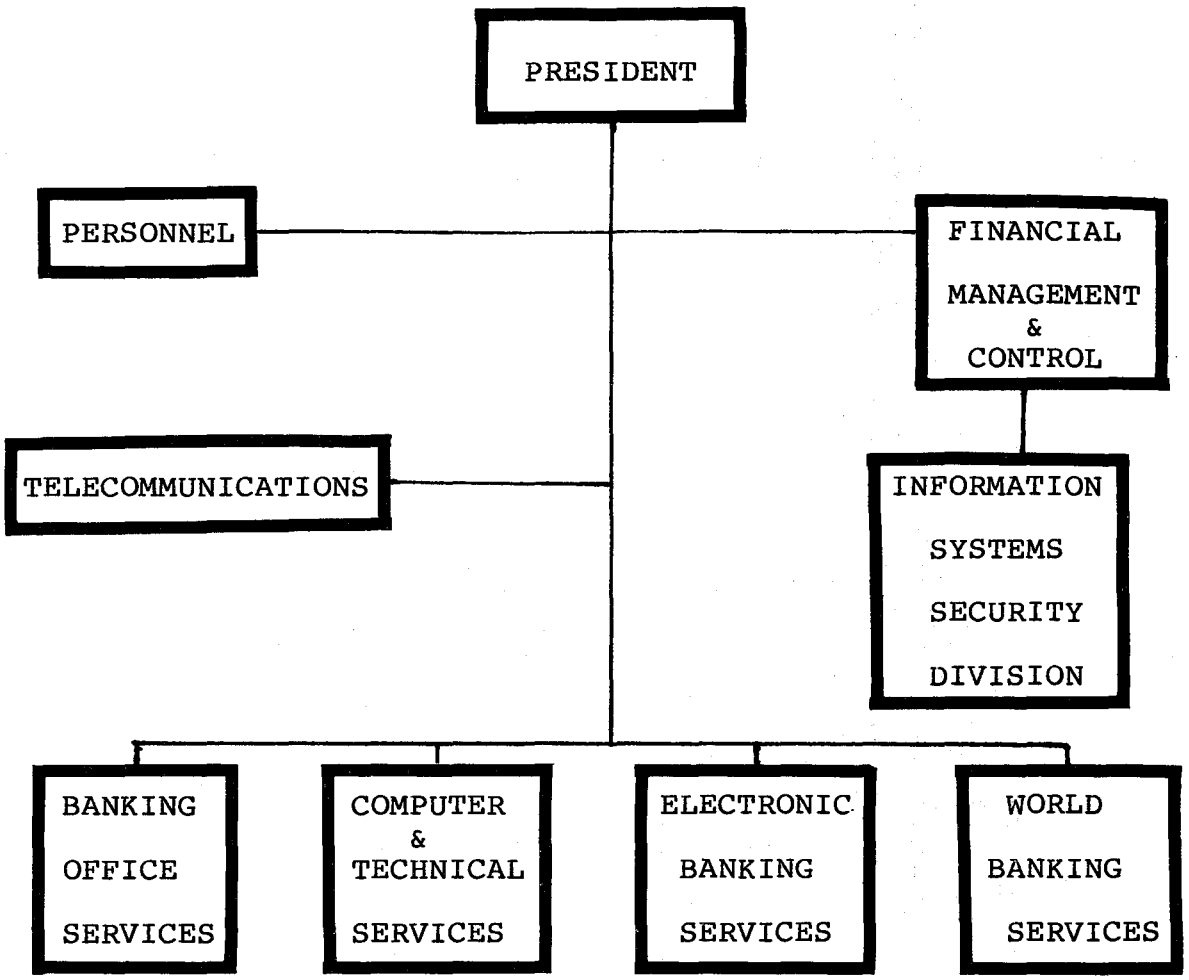
Information Systems Security Division

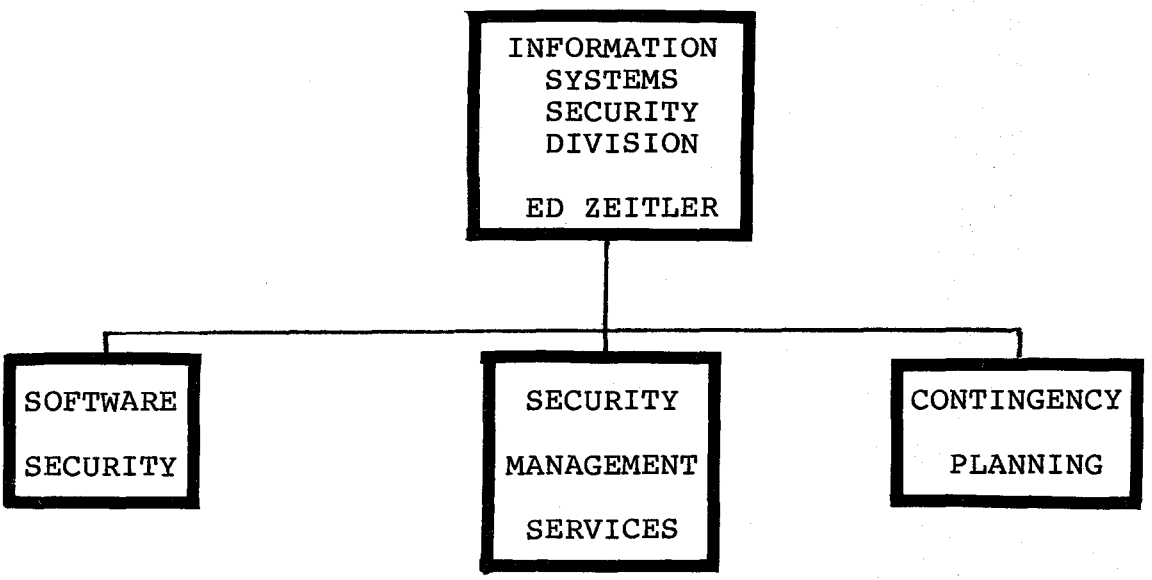
- Organization
- Scope
- Authority
- Strategy
- Responsibilities
- Policy Documentation and User Awareness

Organization

- Major Data Processing Capabilities are Centralized
  - Security Pacific Automation Company
- Information Systems Security Division (ISSD)
  - Staff Function: must not be responsible to a line organization
  - Line Function: must have some operational responsibilities
  - Three Functional Groups
    - Software Security
    - Security Management Services
    - Contingency Planning
- Staff of 20

SECURITY PACIFIC AUTOMATION COMPANY





Scope

- Policy Maker For Bank
  - All Information Systems Issues
    - Information Systems Security Manual (ISSM)
    - User Guides For RACF, IMS, CICS, etc.
    - Specific Procedures
- Control of Access to Mainframes
  - 4 MVS Centers, 1 VM Center
  - Issuance of Userids and Passwords
  - Access to Resources
    - RACF (MVS)
    - DIRMAINT (VM)
  - On-call with Home Terminals
- Control of Cryptographic Materials
  - Key Generation and Distribution
  - Key Management Procedures
- Direct Support
  - SPAC Organizations
    - Contingency Plans
    - Provide Implementation Recommendations or Specific Products for Line Organizations
    - Represent SPAC in Disagreements with Audit Department
  - Consulting
    - Bank Departments
    - Corporate Subsidiaries
- Industry Involvement
  - Industry Committees: BAI, ABA, ANSI, CBCHA, CCTF, etc.
  - Consulting Services

Authority

- Data Processing Security Committee (DPSC)
  - Official Bank Committee
  - Charter
    - Responsible for establishing Policy, Standards, and Guidelines on all matters relevant to the security and privacy of the information processing of customer, employee and corporate information and data.
    - Simple majority required to pass voting issues
  - Membership
    - Audit Department
    - Banking Office
    - Corporate Security
    - SPAC
      - Banking Office Services
      - Computer & Technical Services
      - Electronic Banking Services
      - World Banking Services
      - Telecommunications
      - Information Systems Security Division
  - Authority to Establish Standing Subcommittees
  - Monthly Meetings
  - Provide Avenue for Organizations to Obtain Waivers to Security Policies
- Information Systems Security Division
  - control of critical functions
    - Issuance of Userids and Passwords to Mainframes
    - Control of Cryptographic Materials
  - Limited Formal Authority
    - No Directives: always a business case
    - Prudent, Responsible and Effective Organization

Strategy

- Line Management is responsible for security (each operating center & application).
- Audit Department provides enforcement.
- ISSD provides value added support.
    - Responsible to SPAC Management for the development of sound security policies.
    - Support to Line Management in meeting security policies.

Major Responsibilities

Security Management Services

- Establish Security Policies, Standards, Procedures and Training
- Develop Network Security Measures, including Encryption and Message Authentication
- Manage Cryptographic Keying Materials
- Coordinate Personnel Security Programs
- Perform Physical Security Reviews
- Perform Risk Analyses on Application Development Projects and Recommend Security Measures
- Coordinate Proper Data Processing Insurance Coverage
- Provide Audit Report Tracking and Analyses for Internal and External Audits

Software Security

- Develop Data and Software Security Policies and Procedures
- Develop Access Controls for SPAC Data and Software Resources

- O Administer MVS and VM Access Controls
- O Evaluate and Test New Security Packages
- O Support System and Application Programmers with Security Issues

Contingency Planning

- O Provide direct support to each data center by developing contingency plans which ensure the continuous processing of SPAC's critical applications in the event of a major catastrophe at one of the data centers.
- O Provide assistance with the planning, coordination and evaluation of the testing of contingency plans.
- O Ensure that SPAC Contingency Plans are current by scheduling periodic reviews and plan updates.
- O Provide a consulting service for any bank organization or bank subsidiary requiring expertise or support for data processing contingency planning.

Policy Documentation and User Awareness

- O Primary Documents
  - o Information Systems Security Manual (ISSM)
  - o Contingency Plans
  - o User Guides
- O Information Systems Security Manual
  - o Bank Policy Manual
    - o ISSD writes
    - o DPSC approves
    - o Central Services publishes
  - o Active Document
    - o First publication September 1982
    - o Updates, Additions, and Revisions every Six Months
  - o Contents
    - o General Information and Policies
    - o Responsibilities of Employees and Specific Organizations
    - o Risk Analysis Program

- Personnel Security
    - Sensitive Information Systems positions
    - Personnel Practices
    - Security Education and Training
    - Contractor Personnel

- Physical Security
    - Access Control
    - Fire Protection
    - Electrical Power
    - Air Conditioning
    - Facility Design and Construction
    - Remote Terminals
    - Training, Drills, Maintenance and Testing
    - Data Storage and Media Protection
    - Security of documentation, blank instrument stocks, and other sensitive forms

- Data and Software Security
    - Resource Classification and Ownership
    - Access Control
    - Specific Policies for Online Access Controls in the MVS environment
    - Logging and Audit Trails
    - Violation Reporting and Follow-up

- Network Security
    - Configuration Management
    - Dial-up Controls
    - Message Authentication
    - Encryption
    - Network Contingency Planning

- Contingency Planning
    - Contingency Planning Activities
    - Contingency Plan Maintenance Activities
    - Contingency Plan Activation and Recovery

- Computer Insurance
    - Purchased and Leased Equipment
    - Information Processing Media Insurance
    - Business Interruption Insurance
    - Staff Member Dishonesty
    - Insurance for Errors and Omissions

- Outside Data Processing Services
- Microcomputer Security
- Glossary
- Guidelines & Procedures
    - Waiver Procedures
    - ADP Security Indicent Reporting Procedure
    - Access Control Guidelines
    - Etc.

Contingency Plan

- Introduction and Organization
    - Policy Statement
    - Assumptions
    - Contingency Phases (Notification, Mobilization, Emergency Operations, Recovery)
    - Contingency Organization (list of teams and org. chart)
    - List of Critical Applications
    - Control Group Responsibilities
        - Contingency Manager
        - Management Coordination
        - Damage Assessment
        - Personnel
        - Public Relations
    - User Liaison Coordinator
    - Computer Operations Group Responsibilities
        - Data Entry
        - Computer Operations
        - Offsite Storage
        - Backup Site
    - MICR Operations Group Responsibilities
        - Check Processing
        - Transportation and Logistics
        - Micrographics
        - Printing
    - Technical Support Group Responsibilities
        - Systems Software
        - Applications
        - Data and Voice Communications
        - Data Security
        - Physical Security

Contingency Plan (cont.)

- Notification
  - Notification Procedures
  - Notification Task List
  - Emergency Telephone Numbers (Security, Medica Services, Badge Reader, etc.)
- Mobilization
  - Mobilization Procedures
  - Mobilization Task List
- Emergency Operations
  - Backup Site Preparation
  - Emergency Operations Considerations (Work in Progress, File Reconstruction, etc.)
  - Emergency Operations Task List
  - Emergency Communications
- Recovery
  - Recovery and Restoration Guidelines
  - Recovery Planning Task List
  - Return to Normal Processing Task List
- Appendix
  - Notification List
  - Team Member List
  - Call Sequence
  - Vendor Contact List
  - Contingency Supply Requirements
  - Contingency Kit Guidelines
  - Backup Site Agreements or Contracts

RACF Users Guide

- Overview
  - RACF Environment
  - Verification of Users
    - Profile Structures: Userid and Group
  - Resource Access Protection
    - Data Set Profile
    - General Resource Profile

- o RACF Password Rules
- o RACF Access Authorities
  - o RACF Data Set Access Authorities
  - o RACF Group Authorities
- o RACF Validation During "OPEN"

O Resources to be Protected

- o Production Data
  - o Environmental Data
  - o Control Group Data
- o Non-Production Data
- o RACF Terminal Protection

O How to Protect Data

- o Production Data
  - o RACF Profile Modeling
  - o RACF GDG-Profile Modeling
- o Non-Production Data
  - o Protecting User Data Sets
  - o Permitting Access to User Data Sets
  - o Listing RACF User Data Set Profile
  - o Function of RACF Commands
  - o Protecting VSAM and Non-VSAM Catalogs
  - o Non-VSAM Data Base Data Sets
  - o VSAM Data Sets
- o Accessing Production Data in Batch
  - o Access by Production Jobs
  - o Access by Non-Production Jobs
  - o Userids in Batch
  - o Jobs Submitted From: Local Batch Reader, VM, RJE, TSO
  - o Group Structure
- o Accessing Online Data
  - o Obtaining TSO/IMS/CICS Userids
  - o Program Control Facility II (PCF II)

RACF Users Guide (cont.)

- Revoking and Resuming TSO/IMS/CICS Userids
- Changing RACF Passwords
    - TSO Logon
    - RACF PASSWORD Command
    - Password Changes in Batch
- Trouble Calls
    - S913 Trouble Calls
        - Procedure
        - Responsibilities
        - Documentation
    - Non-S913 Trouble Calls
        - Procedure

User Guide to Online Security for IMS and CICS

- Production IMS/CICS Security Policies
- Online Access Control Facilities
    - Preventive Controls
    - Detective Controls
- Transaction Categories and Required Controls
    - Transaction Categories
    - Generic Transaction Types
    - Controls Required by Risk Level
- Procedures for Resource Protection
    - IMS/CICS Transactions & Transaction Groups
    - CICS DL/I-PSBs and PSB Groups
    - Physical Terminals
    - IMS Logical Terminals
    - IMS Transaction Passwords
    - Non-VSAM Data Base Data Sets
    - VSAM Data Sets
- Procedures for Online Userid Administration
    - Security Administrators
    - User Profiles
    - Obtaining, Revoking and Resuming Production IMS & CICS Userids

- User Terminal Operation
  - IMS & CICS Sign On
    - New and Normal User Sign On
    - Changing Passwords
- Forms Instructions

DEFINITION

"COMPUTER-RELATED FRAUD"

" . . . ANY INTENTIONAL ACT OR SERIES OF ACTS DESIGNED TO DECEIVE OR MISLEAD OTHERS. SUCH ACT MUST IMPACT OR POTENTIALLY IMPACT THE FINANCIAL STATEMENTS AND A COMPUTER SYSTEM MUST BE INVOLVED IN THE PERPETRATION OR COVER-UP OF THE SCHEME."

(SOURCE: AICPA EDP FRAUD REVIEW TASK FORCE)

Jim Watts
General Accounting Office

DEFINITION

"COMPUTER-RELATED FRAUD"

COMPUTER SYSTEM MIGHT BE INVOLVED THROUGH IMPROPER MANIPULATION OF:

- INPUT OR TRANSACTION DATA,
- OUTPUT OR RESULTS,
- APPLICATION PROGRAMS,
- DATA FILES,
- COMPUTER OPERATIONS,
- COMMUNICATIONS, OR
- COMPUTER HARDWARE, SYSTEMS SOFTWARE, OR FIRMWARE.

(SOURCE: AICPA EDP FRAUD REVIEW TASK FORCE)

APPLICATION SYSTEMS AFFECTED

FREQUENCY	BANKING	INSURANCE
HIGH	DEMAND DEPOSITS	ACCIDENT & HEALTH CLAIM
	PROOF AND TRANSIT	LIFE INSURANCE (PREMIUMS, DIVIDENDS, AND SURRENDERS)
	INSTALLMENT LOANS	
	CREDIT CARD LOANS	
	SAVINGS ACCOUNTS	
MEDIUM	COMMERCIAL LOANS	PROPERTY & CASUALTY PREMIUMS
	AUTOMATED TELLER MACHINES	
LOW	CHECK CREDIT	LIFE INSURANCE LOANS
	CASH CONTROL	PROPERTY & CASUALTY CLAIMS
	MORTGAGE LOANS	
	WIRE TRANSFER	

SCHEMES

FREQUENCY	BANKING	INSURANCE
HIGH	CREATION OF FICTITIOUS LOANS	CREATION OF FICTITIOUS CLAIMS
	DIVERSION OF CUSTOMER DEPOSITS	
	DEFERRAL OF POSTING OF CHECKS AND CHARGES	
MEDIUM	EXTENSION OF CREDIT LIMITS	UNAUTHORIZED REFUND OR REDUCTION OF POLICY PREMIUMS
	EXTENSION OF LOAN DUE DATES	
	INTERNAL TRANSFERS BETWEEN CUSTOMER ACCOUNTS	
LOW	FORGERY OF CHECKS	CREATION OF FICTITIOUS LOANS
	EXTRACTIONS FROM ATMS	UNAUTHORIZED DIVIDEND WITHDRAWAL
	ADJUSTMENTS TO DEPOSITS	
	DIVERSION OF LOAN PAYMENTS	
	SEVERAL OTHERS	

PERPETRATORS

FREQUENCY	BANKING	INSURANCE
HIGH	CLERKS (DATA ENTRY, PROOF, OPERATORS, OTHER)	CLERKS (CLAIMS PROCESSOR, POLICY SERVICE, OTHER)
MEDIUM	MANAGERS (LOAN OFFICERS)	SUPERVISORS (CLAIMS, POLICY SERVICE, OTHER)
	DATA PROCESSORS (OPERATORS, SYSTEMS AND APPLICATION PROGRAMMERS)	
LOW	ITEM PROCESSORS	INSURANCE AGENT
	TELLERS	SYSTEMS PROGRAMMER

DOLLAR SIZE VS PERPETRATOR

PERPETRATOR	NUMBER OF CASES BY DOLLAR RANGE (THOUSANDS) UNDER $25	$26-100	$101+	TOTAL
BANKING CASES				
CLERICAL	37	0	1	38
MANAGERS	7	4	6	17
DATA PROCESSORS	9	2	2	13
TELLERS	5	2	1	8
OTHERS	5	2	2	9
INSURANCE CASES				
CLERICAL	17	3	1	21
SUPERVISORS	2	2	5	9
OTHERS	1	2	1	4
TOTAL	83	17	19	119

METHOD OF DETECTION

METHOD	BANKING	INSURANCE	TOTAL
CONTROL AND AUDIT			
INTERNAL CONTROLS	12	10	22
ROUTINE AUDIT	17	4	21
CUSTOMER COMPLAINT/INQUIRY	24	4	28
UNUSUAL OR NON-ROUTINE EVENTS			
ACCIDENT, TIP-OFF, UNUSUAL ACTIVITY OF PERPETRATOR	11	15	26
NON-ROUTINE STUDY	8	1	9
CHANGE IN OPERATIONS, EDP, OR FINANCIAL STATEMENTS	7		7
UNIDENTIFIED	6		6
TOTALS	85	34	119

COMPUTER SECURITY IN PRACTICE

Robert S. Roussey
Arthur Andersen & Co.

The planning committee for the conference has asked me to describe our firm's approach to security and to discuss several case situations. In doing this, I will be using a fairly new medium, computer generated graphics, to help in the presentation.

I will be talking about paths for authorized access to computer assets, as well as discussing unauthorized access. I only wish stopping unauthorized access were as easy as I have now depicted on the screen.

You're all aware of the recent destruction of the log-in and log-out files at the NASA Marshall Space Flight Center and how a hacker was able to penetrate the systems at Sloan Kettering and Los Alamos National Laboratories.

These are just illustrations of some of the key symptoms of the security problem.

And they're symptoms that will grow in importance. For instance, business is placing an increasing dependence on the use of computers for vital operations (sometimes referred to as survival or critical systems); on the use of micros for end-user systems (for communications and for access to the central mainframe); and on the linking of computers with customers and vendors.

This proliferation of micros in business is starting to have an effect on the centralized data center, where information once was generated and stored on a mainframe and was accessible only to a handful of authorized personnel. Now hundreds -- and in some companies even thousands -- of microcomputers are used everywhere. This is the start of the end-user system revolution -- where information, once under centralized control, is now being made available to users, management and others. At Arthur Andersen, we have as many as 2,000 microcomputers in our worldwide organization, and that number will probably double over the next year or so as we implement new worldwide distributed accounting, office automation and professional work station systems. Within five to eight years, we expect that most professional and clerical personnel in our organization will be working in automated environments. Security is, indeed, a concern to us.

The micro is fast becoming a key path to mountains of confidential data. This path needs to be protected in this changing business environment.

While the number of microcomputers now in use is causing concerns, so is the increased power of each new generation of these machines. Over the past few years, phenomenal increases in computer power and decreases in its cost have made it possible for even the smallest company to have computer power that wasn't even in existance only 30 years ago.

As you may know, the classic definition of "balance" in this context is that the cost of security should be less than an expected loss. This expected loss is usually quantified as the Annual Loss Expectancy, or A.L.E. It is calculated as the likelihood of a potential risk occurrence, multiplied by the expected loss from such an occurrence.

Ideally, the annualized cost of security would be less than the A.L.E. by exactly the amount that an organization is willing to accept as a "self-insurance" risk. Practically speaking, however, as you know, both the cost of security and the A.L.E. are difficult to quantify and measure.

For these reasons, the classical approach to using A.L.E. to define the balance point requires considerable tempering and good management judgment whenever quantifiable estimates are not reliable or are subject to change.

We also recognize that there are situations in which "balance" may be a meaningless concept. In many areas of our national defense, for example, there is no price tag for the potential loss in the event of a compromise to security. Also in business, a company may have assets -- trade secret formulas, an industrial process or proposal cost information, for instance, -- that have such great value or importance to the survival of the organization, that even a small probability of loss is difficult to accept. In such situations, the "balance point" may be defined by management edict rather than by attempts to quantify benefits.

Although computer security is a large and complex problem, steps can be taken to approach and maintain proximity to the balance point -- keeping an entity free from unacceptable loss. Such reasonable security is both attainable and desirable.

In our security practice, we seek this balance and address the complexity of computer security in a structured, organized approach to security management.

We have carefully distinguished, for example, this security practice from the normal reviews of security performed on our clients' controls in connection with an audit. The work we do in our security practice is much more extensive and gets more into an analysis of access paths and threat analysis. And we do this work with a practice methodology we refer to as ISSEM. This stands for Information and Systems Security Evaluation Methodology. It is an approach using the concept of an access model, which I will discuss in a moment. The methodology is straightforward and relatively simple, but very powerful. It was developed several years ago. And since then, it has proven to be an excellent analytic tool and has become the basis for our Firmwide Information and Systems Security Practice.

ISSEM is an analytic tool for identifying and evaluating the access paths to an organization's most critical assets. Let me give you a brief overview of this approach.

Consider the data base that contains critical information of some nature -- say, geological and geophysical data of an oil company. Usually you would have at least one program (and often more) that provides access to this information. Let's say that the one shown here is an online update program. You can call up any record on the data base, inspect it and enter changes as needed.

Obviously, any approach to security evaluation would look closely at this process and ask what it can do and who can use it. But there is more. The update process represents the execution of a program or programs that reside in a static form on some program library. There is always some mechanism, a program library maintenance process shown as PLM, for example, that can modify what the program can do. Our evaluation must examine this process.

Also, there is (or at least should be) some reference source that the system can refer to, to determine whether the person using the program is authorized to inspect and make changes to the data base information. Here, I have shown that this may be a file of user ID's and passwords, indicating what access privileges each user has. Again, there must be some mechanism to maintain this information. Thus, both the password file and the maintenence of it are critical parts of our security evaluation.

But don't stop here. The password maintenance process is a program which resides in some library, and which could be modified to compromise the password file. There must be a list of persons who are authorized to enter password maintenance transactions. And anyone who is able to change this list can, thereby, give himself carte blanche access to the critical data base. Likewise, the program library maintenance process has an authorization reference and a static image of the maintenance program, both of which offer potential opportunities to gain unauthorized access to the critical data base.

With ISSEM, an access model of the paths to an asset is prepared. The important points about the ISSEM model are these:

1. The completed model should identify all of the known access paths to the critical data base, which we call the object, at least to the extent this is humanly possible.

2. Each path consists of a series of processes, which may be computer or manual activities, and objects, which may be program code, procedures or data files, and each of these must be addressed in our security evaluation.

3. We must never forget that there will always be unknown access paths to any of the objects. And we must consider the consequence of this at each level in the model.

Now, let's see how the access model concept is applied.

Our ISSEM methodology consists of the following six major tasks:

- Building an access model for each key object we evaluate,
- Determining the control objectives for each process and object,
- Identifying the potential risks,
- Identifying the existing control techniques that may mitigate the risks,
- Rating each risk on a scale of one to ten, and
- Developing scenarios showing how each risk may be exploited.

Let's go back through those six steps and look at them in detail.

The access model begins with a key object to be protected, for example the data base of critical information mentioned earlier. Since that data base is passive, we contend that all access to it must be through some process. Therefore, our key is to find all the processes that can access that particular object.

For each process, for example as shown for Process A, we identify several additional objects; here we have labeled those as the subject, the resource and the authority reference.

The subject, at the bottom, may be a source transaction or transactions passed from another process. The term "transaction" is used in its broadest sense -- meaning any event that initiates the process. This may be a transaction entered at a terminal, or it may be a data file from another system.

The resource is typically a program or procedure used in the execution of a process. One way to gain access to the data base is to modify one of the resources used by the process.

The authority reference represents any measures that may be in place to determine if the subject is permitted to use the resource to access the object. It also determines the conditions and type of access that are authorized. For example, the time of day access is permitted, the areas of the data base that can be accessed, and what can be done to the contents of the data base --modify, destroy, etc.

While the access model may seem relatively simple as I've described it, it can become quite complicated. Each process may have multiple subjects, resources and authority references. And each of these new objects can be accessed by one or more other processes. The result is a repeating pattern in the access model. Authority reference A is accessed by process B. Process B has a subject, authority reference and resource. Similarly, resource A is accessed by process C, which also has an authority reference, resources and subjects.

This repetition continues until we reach a <u>trusted</u> object, which usually includes people, policy documents and objects that are controlled by systems beyond the scope of our evaluation.

The purpose of drawing this access model is to identify each access path, which is simply the route from an entry point at the bottom of the access model up to the key object. In reviewing the paths, the key evaluation question is: "Are there ways to use each path in an unauthorized manner?" We want to be certain that all of the risks identified along the path are also controlled. We also want to remember the possibility someone will find access paths that we don't know about. Thus, we consider the implications of this for each object in the model.

The second major task of ISSEM is applying control objectives to the process. Control objectives are our way of expressing management's decisions on how the business and systems should operate, and they become the basis for our evaluation of the actual system.

Typical control objectives might be that:

- The program library will be modified only by the librarian with approval of the acceptance test group supervisor, and

- Salary information will be disclosed only to the head of the department and the assigned personnel manager.

When we have identified and applied management's control objectives, we begin the third step, identifying the potential risks. We define a potential risk, by the way, as a violation of a control objective and not as a business consequence. For example, a risk exists if the program library can be modified by the librarian without the authorization to do so, or if it can be modified by someone other than the librarian, or if authorized changes are not applied.

Our fourth step, then, is to identify the existing control techniques that may mitigate the risks we have identified. Control techniques -- sometimes called safeguards -- are mechanisms that deter, prevent or detect and recover from risks. They may be passive or active, manual or automated.

In step five, we rate the risks on a scale of one to ten. We do this by relating these control techniques to risks through a matrix. This matrix primarily serves as a way of double-checking that all risks and controls have been identified.

Risks are listed as row headings on the matrix, and control techniques as column headings. When a control is related to a risk, an X is placed in the intersecting box. This is not a control evaluation, only the identification of some relationship. A row with no X indicates an uncontrolled risk, or a control technique that has been overlooked. A column with no X indicates an unneeded control, or a risk that has been overlooked, which tells us we have more work to do.

Each risk is then assigned a rating, with ten being a high risk. Risks rated above the threshold -- usually set at four -- are called exposures.

For each exposure, we develop a scenario showing how it may be exploited. This is the sixth step in our ISSEM methodology. The purpose of these scenarios is to communicate the exposures to management and provide other information that will assist management in making judgments on the additional means of confirming that the referenced exposures are, in fact, not under control.

Information provided in a scenario includes:

- The exposure or exposures upon which the scenario is based,
- Resources required to exploit the exposure,
- The number of people who are potential perpetrators. An exposure that can be exploited by 200 people is obviously of greater concern than one that can be exploited by one, two or three.
- The motive, which is usually personal gain or gain to the organization,
- The procedures the perpetrator would need to follow,
- Existing controls that might catch the execution of this scenario, and finally,
- The extent of damage to the organization that would occur before the scenario would be stopped.

As you can see from these steps I've just talked about, ISSEM takes a much more rigorous approach than many "checklist-based" types of evaluation; ISSEM

not only identifies what controls are in place, but also focuses on how a knowledgeable person could circumvent these controls. And the evaluation project team usually feels that ISSEM has identified potential risks that they might otherwise have missed.

ISSEM also improves management's understanding of computer security. In doing so, it helps them understand where they stand in relation to the ideal balance between over-control and under-control...between spending too much for security and not enough to assure the protection they need.

Now, let's look at two specific cases in which we've used ISSEM to evaluate the effectiveness of clients' security procedures.

The first one is a medium-sized European bank that is heavily involved in international currency markets. The bank was extending the scope and features of its recently computerized transaction processing and reporting functions. We were asked to evaluate the security of the bank's information systems.

First, we conducted a general risk analysis in which we reviewed the overall operations of the computer system and identified areas of high potential exposure. Then, using ISSEM, we performed a detailed evaluation of selected exposure areas.

In the general risk analysis, which took about two weeks to complete, we identified some of the key exposures as follows:

- o The computer files relating to the bank's international currency trades represents a high potential exposure. There were indications that the bank could be subject to serious disruption and financial loss from potential violation of the system by knowledgeable individuals.

- o Systems documentation deficiencies resulted in undue reliance on one data processing manager. This situation made the data processing department potentially susceptible to disruption if this manager were to resign. And it was further aggravated by the fact that an adequate set of data processing standards and procedures had not been established.

- o Next, we found that a disaster recovery plan needed to be formalized and parts of it tested as appropriate. Although not likely to occur, a major disruption to the computer facility would have had a significant effect on the bank if such a plan were not available at the time of disruption.

- o On the positive side, however, we discovered that the bank's overall controls over the computer system were good. We also found that the computer environment and access to it appeared well controlled.

After identifying these general risks, we started the more detailed ISSEM review, which took about 15 weeks for completion, and included an evaluation of the bank's currency trading data base and system, the accounts file and the back-up files in the computer systems. In performing the work, we developed and documented ways that these key assets could be compromised by a (hypothetical) person who might know the bank and its systems. We reviewed and documented management controls over these files, and evaluated the resulting levels of security. We classified the resulting security exposures according to potential materiality, potential for occurrence and effectiveness of the controls.

Among our findings and recommendations were two that we thought were most important:

o Improve password administration, which we found was weak. The bank was conscious of the need for adequate segregation of duties in structuring their organization and assigning responsibilities, but had not recognized that this important control had been compromised when they assigned user ID's and access privileges based on functions rather than individuals. They also lost some employee accountability because of the sharing of user ID's and passwords. Other related problems were also identified.

o Restrict the programmers' access to production programs and documentation. Programmers were in a position to learn about details of the system's user interfaces and controls, could access and modify the production programs, and had opportunities to operate the computer.

The other assignment I want to look at briefly is a major commercial bank that has on-line systems serving 4,000 terminals and automatic teller machines. The officers of the bank felt that security was good because thay had always maintained a strong emphasis on security. However, they had some concerns that the checklist reviews they were using might not be providing a high enough level of security assurance. Thus, they wanted an independent firm to take a very exhaustive look at security over key assets. Specifically, they were concerned that their checklist reviews didn't provide the desired level of assurance.

We did not conduct a general risk analysis because bank management had identified the customer account data bases as the critical assets to be evaluated.

Our assignment was to apply our ISSEM methodology to these data bases. During our evaluation, we found a number of major security problems and suggested procedures for eliminating them. Some of these were:

o The bank had implemented effective controls over transactions originating in the branch offices. These controls included access control software, segregation of duties, dual authorization for high dollar transactions, extensive control logging and exception reporting, and review requirements. Their systems and operations group, however, could enter transactions without proper authorization, and the logging and review of their on-line activities were weak. In particular, the lack of adequate segregation of duties between systems programming and the data base administration function created a material vulnerability for the entry of unauthorized transactions.

o Certain functional areas of the systems and operations group also had an unwarranted degree of free access to system-level command transactions that could permit them to reassign terminals authorized for specific locations, or even to modify entries in the security software authorization tables. The bank's auditing procedures would probably detect such unauthorized activities, but not until some days after it occurred. This time period might be critical to a fraudulent scheme.

o Computer employees generally operated under "carte blanche" procedures in the computer room. Operators had free access to all production and systems program libraries and to input/output control areas. Controls originally established to provide segregation of duties and access

authorization controls in this department had deteriorated to an extent that operations represented a significant security exposure.

What have we learned from the application of ISSEM in a wide variety of environments, such as those I've just described? First, we've learned that there are about five significant exposures that we frequently encounter.

The "emergency quick-fix" typically provides a procedure for operators or programmers to apply "patches on the fly" to expedite a production run, often on the third shift late at night. The weakness with this exposure is the company's dependence on the operator/programmer to report the "fix." Usually he doesn't.

The rapid growth of microprocessors has resulted in micros being connected directly to the mainframe for the convenience of vendor maintenance and systems or application programmers. These units have, in some cases we have seen, been set up as if they were operator consoles. They were in effect privileged terminals with ready access to production processing and files. In addition, remote access using microcomputers is a growing concern.

The use of passwords is one of the most misunderstood control techniques. Without question, the most frequent exposure we've found has been the improper use and control over passwords. Segregation of duties normally found outside the use of computers is often compromised by allowing certain users to perform all functions with their passwords, thus compromising what appears to be a well-controlled environment.

Having no real contingency plans is a problem familiar to most of us. Even when contingency plans have been developed, we find in many cases that they are not maintained and, therefore, are likely to be useless if and when they're needed.

Programmer access is also a familiar problem. There continues to be the feeling that programmers need access to everything in the system to get their jobs done. But, in fact, there's no good reason not to implement the normal segregation of duties between the systems programmers, payroll programmers, financial reporting programmers and so on through the security features of the system.

We've also learned that any organization that has seriously addressed its security needs is still likely to have a risk profile that includes both high-level and low-level risks. In each engagement, we attempt to determine the level of risk by applying the risk rating procedure I described earlier to evaluate the materiality of loss and the likelihood of an occurrence. The chart on the screen was developed from a composite of about 2,000 risks taken from about 20 engagements. First, it shows that a company is likely to have a large number of exposures that are regarded as low-level risks. It also shows that a company is likely to have a few high-level exposures but were not adequately controlled. In effect, about 16% of all risks identified fall into the high-level exposure area.

Thus, we've learned that a few risks at the high end of the scale may go undetected and their potential exposure unrecognized unless a rigorous, analytic approach to security management is used. Even then there will be some risks that are not found. Such an approach, however, provides greater confidence that most of the risks are known and that security measures are in place to prevent their occurrence.

A COMMERCIAL USER'S PERSPECTIVE
(With A Proposal)

James A. Schweitzer

INTRODUCTION

Current computer uses are usually driven by economics. That is, computers provide ways to accomplish tasks at an overall cost significantly less than those for completely manual methods. The people using the computers are not, typically, computer literate. They view the computer as a tool (like a typewriter). The security for these applications is not a constant, but relies on the risk-acceptance posture of business management, expressed in terms of security funding.

Security vulnerabilities are created, in large part, because the human-system interface protocol is infinitely variable across applications, and considering the users' motivations actually discourages good security practices. Since a very large portion of the abusers are authorized users, the failure to provide an effective, supportive interface protocol is a direct cause of many, if not most, computer abuses.

A solution, in part, appears to lie in improved administration, and in provision of high quality, standardized user-machine interface protocols. These are described in detail.

I. Background

Almost all commercial computer applications are driven by economics. That is, the fast-reducing cost of a MIPS is increasingly attractive against the total wage costs of administrative or other knowledge workers. Computers are used as a cost-beneficial replacement for manual work (consider word processing as an example).

In many instances observed, a non-computer literate employee is using more than one system. A general clerical employee in a "branch" or "store", for example, may use several systems with unique protocols:

- word processor
- network communications
- customer administration
- inventory/order status
- service request control.

A wide and perplexing variety of user interface protocols may occur across such a mix of applications. For example:

User/system Process	Application (illustration only) 1	2	3	4	5
sign on	i.d. only	org. & i.d.	funct. key	password	none
identification	acct. no.	acct. no.	i.d.	password*	none
authentication	file name[x]	password	none	file name[x]*	file name*
expiry reminder	no	yes	no	no	no
expiry cut off	no	yes	no	yes*	no
monitor/log	no	yes*	no	yes*	no
password const.	none	1-6 a/n	none	4-8 n	none
lock word const.	none	1-8 a/n	none	1-8 a/n	1-8 a/n*
pw encrypted	n/a	yes	n/a	no	n/a
error recovery	user	local s.o.	user	central s.o.	user

* optional
[x] file name is lockword if that security feature is used

The selection of hardware/software is usually application relevant. That is, the management selects from the system components offered those which best interface with existing data structures, communications networks, and/or practical use requirements. Security is (seems to be) always a subordinate requirement. The current array of product (hardware and software) offerings provides a staggering variety of user interfaces, even within a single manufacturer's line.

Complicating the issue is the tendency of profit-oriented (properly so!) managers to state a security policy and then display a risk acceptance posture at wide variance with the policy. This display consists of spending authority decisions. Usually, the security elements provided by the computer/software supplier are accepted per se; seldom, if ever, is supplementary funding available for providing a security supportive user interface for an application. The bewildering variety of interface protocols thus find their way into the business operational situation.

II. Vulnerabilities

Current systems applications present two general categories of vulnerabilities, convenient for our purposes. These are:

1. Hands-on vulnerabilities, or those which involve direct access to a component of the application system, e.g., a single purpose terminal, an application-directed microcomputer, a specialized device (a card-reader or mark-sense reader?) or the mainframe computer supporting the application.

2. Network connected vulnerabilities, or those occurring through accesses via general purpose, connected devices. Note that these connections may be authorized or unauthorized (penetrations).

Under Category 1 (Hands-on), experience in a commercial environment indicates that almost all security violations are committed by employees or recently discharged employees who have (had) authorized system access. In some cases the violations involve collusion with persons inside or in related (e.g., suppliers, consultants) outside positions.

Under Category 2 (Network connected) the violations are mostly authorized employees. The other cases authorized "outsiders" (e.g., supplier people given access rights for special purposes such as supplied parts quality control). Also in that grouping are unauthorized outsiders ("penetrators") who gain access through clandestine means. Still another category of increasing concern, as network interconnects grow, e.g., ARPA, are the "gray users"; these people may have access for apparently innocuous purposes, but may develop into sources for proprietary information.

Some cases, based on actual experience or professional contacts (names changed to protect the guilty), will serve to illustrate the vulnerabilities.

a. A discharged senior programmer was allowed to return, unaccompanied, to the work place where the programmer secretly changed the passwords for programmer's system access.
Analysis: Administrative failure.

b. A data entry clerk arranged with an accomplice at a supplier to falsely process, and then destroy, evidence of purchase/payment transactions.
Analysis: inadequate system audit trails, failure to review transactions.

c. A hackers' network provided information for access to a business computer system. The information was correct although only partially complete.
Analysis: Perception of administrative difficulty in coping with change of password made "old" data still useable.

d. An engineer was able to re-create proprietary product design data by "browsing" through network-connected files. Normal security identification for the data summary was not present in such a bits-and-pieces compilation.
Analysis: System failed to establish protection; assigning restrictions during normal use was inconvenient. Security mechanisms inadequate.

e. A scientist reported that files had been erased.
Analysis: Password had not been changed for years.

An analysis of these and many other cases, publicly announced and privately told, indicates that the causes of security failure are (in order of frequency/effect):

1. Administrative/user carelessness
2. Inadequate system support for security
3. Weak or irrelevant system security elements

III. Recommendations

The tertiary cause of the security failures noted above is being actively addressed by computing professionals, as witness the symposia of the IEEE Technical Committee on Security and Privacy, the DoD Computer Security Center, and the ACM Special Interest Group - Security, Audit and Control.

This paper addresses the primary and secondary causes, (1) Administrative Controls and User Motivation and (2) System User Interface Protocols.

Administrative Control and User Motivation

Beautifully constructed and artfully programmed systems, implemented through hardware of breathtaking performance, are commonplace. Unfortunately, the traditional control elements are often omitted. These elements should include:

- effective audit trails to allow proving of the authenticity and correctness of system activities. Note: this is true of any system, as the integrity of data relies upon proper (read "authorized" or "responsible") system use.

- redundant checks where fiscal elements are involved, e.g., procurement, payment, personnel/payroll, etc. Before automation these typical "accounting" separation of duty practices were taken for granted.

- management control of user authorizations for system use. This includes a monitoring function which reviews system activity logs. The concept of information resource management (IRM) is essential to provide authoritative decisions on data values (classification) and/or system user rights (to see, move, modify, write, run, etc.). This is a frequent failure; many functional managers believe the "data belongs to the data center manager."

- Security control over access rights and the timely implementation of IRM decisions on access privileges (usually job related). Cancellation of rights and motivation of security discipline and practice fall within this responsibility.

System Support for Security

A recommendation for systems support is provided in the paper "A Proposal for an Automated Logical Access Control Standard" (January, 1984) which follows.

LOGICAL ACCESS CONTROL EFFECTIVENESS

Standard commercial logical access control software designed to protect business data normally puts three steps in the way of the User who wants to sign-on to the computer and perform certain tasks. These are:

(a) **Identification.** The User is required to enter a valid and uniquely identifying code, or "UserID." This could be, for example, a personnel number; it commonly doubles as an account code.

(b) **Authentication.** Having been identified, the User is required to provide some code or token which is privately known or personally

held, to authenticate that the User is really who he or she claims to be. This could be a plastic card, fingerprint, or recognizable voice pattern. However, by far the most common authentication means is for the User to enter a private password. From here on we shall consider only passwords as authentication means. Other forms of token may become common in the future, but that will not change the validity of our position on passwords.

(c) **Authorization.** The authenticated User is then permitted to perform only those actions, e.g., access and update certain files, execute certain programs or transactions, read certain documents, etc., which have been pre-authorized.

To examine the security effectiveness of logical access control mechanisms in preventing unauthorized access, we must again consider these three logical steps independently.

Firstly, the UserID has a limited security role. If a personnel code such as Employee Number is chosen to be used for the UserID, then such codes may be general knowledge within the organization. UserID's are often required to be entered in clear text and can therefore be seen by passers-by, or obtained from printout. If doubling as an account code they will appear on invoices for computer usage which may pass through many hands. UserID's which become known to employees who leave the business with the knowledge, as well as contract staff, visitors, etc., effectively become public knowledge. Thus, although it serves to distinguish each individual who may use the computer, a UserID may have no security value at all; it represents (only) a "claim to be" a certain person.

The authenticating password is a different matter. If the password is properly constructed, and if the User keeps it secret and private, not easily guessable, and changes it periodically, then a password can provide a highly effective security mechanism. The crucial word, of course, is "if". It is a matter of common knowledge within large data processing installations, confirmed by published articles in computer journals (Ref. 11), and by newspaper stories of computer "break-ins" (Ref. 2), that User password discipline is often poor. Passwords are written down, easily guessable strings or names associated with the User are chosen. Passwords are seldom changed, and then only as a result of enforcement exercises by the security or data processing management.

The third logical element, authorization processes, are usually well designed and enforceable. Time-sharing systems, for example, will commonly allow an authenticated User to access only "public" program libraries, or data or programs which he/she has personally entered or created. Extension of access rights to other Users requires some positive

action on the part of the data owner, or of a security administrator. Likewise, properly designed transaction-processing systems can be made completely inaccessible to a given UserID unless that UserID has been specifically authorized to execute certain transactions by a security administrator. Authorization mechanisms have received the greatest attention by designers of security systems. For example, the U.S. Department of Defense "Trusted Computer System Evaluation Criteria" (Ref. 3) concentrates very heavily on authorization mechanisms. Generally speaking, therefore, at reasonably low administrative overhead cost, authorization rights to access data can be automatically enforced, and do not interfere unduly with business data processing needs.

It is obvious from the foregoing that the weak link in the chain of logical access security is the password. Users of business data processing services are motivated much more by their desire to get on with their computing, than by a concern for data security (generally users have a low perception of risk consistent with management views); password discipline is not normally enforced, and hence password discipline is poor.

Attempting to enforce password discipline by exhortation or supervisory methods is ineffective. In today's business conditions the policing of a large population of Users with terminals at home and elsewhere is not practicable. Such effort is in any case unpopular - it smacks of bureaucracy.

Conclusion: This analysis and train of reasoning lead to the idea of developing a specification which would require the computer itself to enforce logical access security, automatically as fas as possible, at each stage of use from sign-on to sign-off. The specification should pay particular attention to enforcing password discipline.

ALACS - AUTOMATED LOGICAL ACCESS CONTROL STANDARD

ALACS has as its objective to specify an optimum set of logical access controls in order to provide adequate security for typical confidential business data; these controls are to be automatically enforced by the computer's operating system, and if necessary, supplemented by the application system. Where not automatically enforceable, because of system limitations, ALACS should provide maximum computer assistance for the manual administration of security.

A full specification of ALACS is given in the Appendix to this paper, and the security objective for each requirement is explained. ALACS suggests a standard iterative sequence of steps as a "log-on" process. The process occurs each time a user attempts computer activity.

A set of security design principles was defined by Hoffman (Ref. 5). The ALACS proposal meets these principles, especially those marked *.

1. Default to access denial
*2. Non Secret design
*3. User Acceptability
4. Complete mediation
5. Least privilege
*6. Economy of mechanism
7. Separation of privilege
8. Least common mechanism

It should be emphasized that ALACS contains no new security features which have not at some time been proposed and/or implemented by some computer supplier. Many of the requirements for passwords are described in a FIPS publication (Ref. 4). However, a survey of the logical access controls of 12 software systems from 7 unique combinations of leading hardware/software suppliers showed that none met ALACS completely, and most had several significant weaknesses.

The novelty of ALACS is only that it covers all aspects of logical access control requirements for typical business data in one statement, and that it specifies a sign-on protocol which maximizes automatic enforcement of password discipline in a User-friendly way. The parts of ALACS concerned with access authorization mechanisms are already standard practice on many suppliers' computer systems.

Publishing ALACS as a proposed standard serves two goals:

1. In the short term, an organization can compare its existing computer logical access control mechanisms with ALACS as a means of identifying weaknesses in security effectiveness. These weaknesses can then be rectified by in-house software modifications and/or brought to the computer supplier's attention; alternatively the risk of the weakness can be consciously accepted. At best, security is improved; at worst, awareness of security vulnerability is heightened.

2. In the longer-term, ALACS should stimulate debate in the business data processing community, and especially standardization bodies. The debate will no doubt produce suggestions for improvements to ALACS. With increased customer pressure and public concern arising from the computer trespassing threat, computer manufacturers and suppliers of security packages should find it worthwhile to implement ALACS fully.

A full and uniform implementation of ALACS would not only bring a great improvement in computer security over the current level, it would generally help make computers easier to use. Today each business computer system has its own unique sign-on procedures and security mechanism. Imagine the parallel if every time you took a hire car it was necessary to get out a manual to work out how to start the engine and drive off. Car manufacturers have evolved a standard user-friendly man-machine interface, and increasingly that interface incorporates safety-enforcement mechanisms (such as reminders about seat-belt wearing), - analogous to ALACS' requiring computers to enforce security. Although "standard" in a conceptual sense however, each car's interface is realized in practice in a unique way as far as detailed presentation and aesthetics is concerned. Likewise, ALACS is a conceptual specification. Details of presentation are left to the implementor.

A standard such as ALACS, therefore, has wider implications than security. The improvements in computer-user acceptability and productivity which would follow widescale implementation of ALACS, especially for large organizations using computers from many suppliers, will in the long-term be as valuable as its security benefit.

Reference

1. "Password Security: A Case History", Morris R., and Thompson K., Communications of ACM, 22, no. 11, November, 1979.

2. "Trial and Error by Intruders Led to Entry into Computers", New York Times, August 23rd, 1983.

3. "Trusted Computer System Evaluation Criteria", US Department of Defense Security Centre, May 24th, 1982.

4. "Guideline on User Authentication Techniques for Computer Network Access Control", FIPS (Federal Information Processing Standards) Publication, PUB 83, 1980 September 29.

5. MODERN METHODS FOR COMPUTER SECURITY AND PRIVACY, Lana J. Hoffman; Prentice Hall, 1977; pp. 3 and 4.

6. ALACS Proposal, attached. Charles R. Symons and James A. Schweitzer.

Appendix: Automated Logical Access Control Standard (ALACS)

Charles R. Symons
Nolan, Norton & Co., London

James A. Schweitzer
Xerox Corporation

Objective

To specify an optimum set of logical processes which can be implemented on a computer to control access to confidential data and text held on the computer for the purposes of maintaining adequate security.

Scope

- Any computer whose use is shared by a closed community of Users, any of whom may use the computer and/or access files via terminals, microcomputers, etc., for certain pre-authorized purposes,

- Where the data or text which are held are either

  - confidential to the organization or part of the organization using the computer and/or

  - subject to other need-to-know restrictions, e.g., for internal control reasons, personal privacy, etc.

Terminology

In this standard the word "computer" is used to encompass any computer, system or sub-system, shared intelligent workstation, or group or network thereof, with which a User communicates, in interactive or batch mode, and which is a separate entity from a security control viewpoint. The controls described within ALACS may be distributed over various processors within the "computer", as appropriate to its configuration and security needs.

Requirements

The following minimum requirements must be met. For each requirement the corresponding security objective is given alongside.

	Requirement	Security Objective
A	**UserID.** A UserID, minimum six characters, must be assigned to each individual User, which is unique to that computer. The computer will not allow two or more terminals to be signed on simultaneously with the same UserID.	Inhibits sharing of UserID's, and emphasizes individual accountability for usage and security.
	Although assigned to an individual person, a UserID may belong to one or more recognized groups of UserID's which share common access authorizations. (See para. C.2.a below.)	Helps simplify administration of access authorizations.
B	**Passwords.** Each individual UserID must have an associated password which the User is instructed to keep private with the following characteristics:	Password is the key to authenticating that the User is indeed the individual identified by the UserID.
	• **Length.** Minimum of 6 alpha-numeric or special characters, excluding blanks.	Makes password harder to guess by trial-and-error or to discover from systematic testing.
	• **Frequency of change.** The computer will force a password to be changed within D days of the last change, where D is an installation parameter with maximum 99 days, default 30 days.	Forced password changing reduces the security exposure if an existing password has become known to other persons than the password-owner. Forced changing also heightens general User security consciousness.
	• **Repeatability.** The computer will maintain a list of the last P passwords used by the UserID and will not accept an attempt to change to a password already used and still in the list. P is an installation parameter with a minimum of 10 passwords.	Inhibits the User trying to beat the enforced password changing control.
	• **Initialization.** When a new UserID is established it will be given an "expired" password (see C.1.c below), that is one which must be changed at the first attempted sign-on by the UserID.	Prevents the person allocating UserID's from knowing the password which will be used by the User concerned.

- **Encryption.** All passwords will be stored in the computer in one-way encrypted form. A password entered during an interactive sign-on, or a batch job submission will be immediately encrypted at the time of entry, and thereafter never displayed in clear text.

  Prevents a system programmer or someone working in "privileged" mode (see C.3 below) from obtaining passwords and thereby being able to impersonate any UserID.

C **Logical Access Control**

1. **Sign-on (Identification/Authentication) Phase**
   Sign-on will follow the procedure below, from the point where the computer is ready to accept identification of the User via a UserID.

   a. Computer invites sign-on by requesting entry of the UserID, in an indicated field. If accepted the computer proceeds with step b. If not accepted the computer allows up to two more attempted entries, and then if still unsuccessful:
   - logs all unsuccessfully tried UserID's,
   - alerts Operator or System Security Administrator
   - (if appropriate) disconnects the terminal.

   Procedure is designed to help the genuine User, but inhibit someone trying to find an acceptable UserID by trial-and-error.

   b. Computer invites entry of password, in an indicated field, but provides a "blot" (or inhibits display or printing) for that field so that the entered password cannot be read. User enters password, and if successful, the computer proceeds with step c. If unsuccessful the computer allows up to two more attempted entries, and if still unsuccessful:

   - logs all unsuccessfully tried passwords
   - alerts Operator or System Security Administrator
   - (if appropriate) disconnects the terminal.

   Procedure is designed to help the genuine User, but inhibit a casual observer from seeing the password, or someone trying to guess a password by trial-and-error.

N.B. The computer should enforce a time delay of minimum two seconds between repeated attempted entries of a password.

Inhibits someone successfully using a computer to generate passwords systematically to gain entry.

c. The computer checks if today the password is more than E days from the date of expiry (E is an installation parameter, usually set to 20% of the forced change period D). If the password is still more than E days from the expiry, the computer proceeds with step f.

To be as helpful as possible the computer gives advance warning to a User whose password is due to expire imminently.

d. If the password is within E days of expiry, but still unexpired, the computer issues a warning giving the number of days remaining before the password must be changed. Alternatively, if the password expires today, or is already expired, the computer informs the User that the password must be changed immediately.

e. The computer issues an invitation to change the password, indicating the format, and supplying a "blot" (or inhibiting display or printing). The User may ignore the invitation to change by pressing "Return" unless the password is already expired, or expires today. If the User enters a new password, the computer invites a repeat entry to validate the first entry (similarly concealed), and continues until two successive identical passwords are entered.

The computer helps the User change password, and enforces change of an expired password. A changed password is requested a second time to avoid problems which would be caused by a typing error during the first entry, and to reinforce the new password in the User's memory.

f. The computer issues a message stating the date and time when the last successful sign-on was made.

Provides a check for the User that his UserID has not been used without the User's knowledge.

Batch Job or message submission from an interactive terminal or workstation

g. The computer will only allow a batch job or message to be submitted for execution, or sent from an interactive terminal or workstation if the batch job or message is associated with the same UserID/password combination used for initial sign-on.

Prevents a user signing on under one UserID with associated authorizations and then creating and submitting a job with a different authorization.

h. Sign-on proceeds essentially as in Interactive mode, except that the computer does not provide guiding messages, and if any step is unsuccessful the job is cancelled, with the appropriate explanation

2. **Processing (Authorization) Phase**

a. Any computer to which the UserID may gain access will control, using information provided by the owner of the "object" concerned:

Each UserID should be limited in what use can be made of the computer by pre-agreed "need-to-know" considerations.

- The list of "objects" (programs, transactions, files, etc.) to which the UserID is allowed access either individually or by virtue of membership of a recognized group, or of pre-registered attributes.

- The "level" of access (read, copy, update, create/delete, execute) allowed to the objects.

Additionally, the computer will warn the User (interactive mode) or cancel the job (batch mode), if the User tries to access beyond the authorized range or levels.

b. The list of UserID's, or recognized groups of UserID's, which may access any object, and the associated level of access may only be changed by

- the UserID which individually created the object, or
- the object's "owner" (if such is established), or
- the System Security Administrator working in "Privileged Access" mode (see below).

The rules and mechanisms for changing access authorizations must be clearly and coherently established; they will vary depending on the type of computing service. Time-sharing and office systems usually allow only the creator of an object to change the access authorizations. In contrast, a community of Users sharing a common data base is better regulated via a System Security Administrator acting on behalf of the data base "owner."

c. Any major sub-system executing on the computer which is shared by Users with different "need-to-know" requirements, and which is treated as a single "object" by the computer's security system, must itself provide its own authorization scheme along the lines of a. above.

The computer's security system may not be able to cope with incompatible security conventions of a "foreign" subsystem. The latter must therefore provide its own authorization mechanisms.

d. If a terminal or workstation is inactive for more than T minutes, the associated UserID will be automatically signed-off. T will be an installation parameter with a default of 15 minutes.

Prevents someone using a terminal which has been left by a User who forgot to sign-off.

As an alternative to sign-off, the computer blanks the terminal screen and requires re-entry of the User's password to resume the session.

Alternative caters for the case where the overhead due to sign-on/sign-off is unacceptable.

3. Privileged Access

A privileged access mode will be available to a System Security Administrator for maintenance of all security and logical access control parameters, but only for those purposes. Privileged access will not be needed for any application programming, or use of an application or utility program.

A privileged access mode is essential for security administration, such as establishing and deleting UserID's, changing certain types of access authorizations, etc. Such a privileged access mode must itself be protected from unauthorized use to at least the ALACS standard.

4. Logging

All unsuccessful sign-on attempts, and all unsuccessful access attempts during processing (both of "range" and "levels") will be recorded in a log in the computer concerned, available only in privileged access mode. All log message-types will be uniquely coded, and date and time stamped to enable analysis. Analysis programs will be provided which highlight suspicious repeatedly unsuccessful sign-on or access attempts.

A log of attempted security violations is an essential defense mechanism to help a System Security Administrator discover apparent deliberate attempted violations.

5. Authorization Maintenance

Administrative Procedures will be established for each computer such that

- if an individual leaves the organization any individual UserID is immediately cancelled.
- if an individual's job is changed, then any consequential changes of the individual's authorization to access programs, transactions, data, etc., are immediately effected.

Sound procedures to administer UserID's are an essential counterpart to the computer-enforceable security measures.

D **Optional Refinements**

1. **Physical Terminal constrained to certain UserID's.** The computer may allow only certain UserID's to sign-on to certain physical terminals.

   This is a valuable option for situations where specific computer processing should be possible only from certain terminals which could be at a specific secure location, equipped with certain security features, etc., due to the need to handle particularly sensitive data.

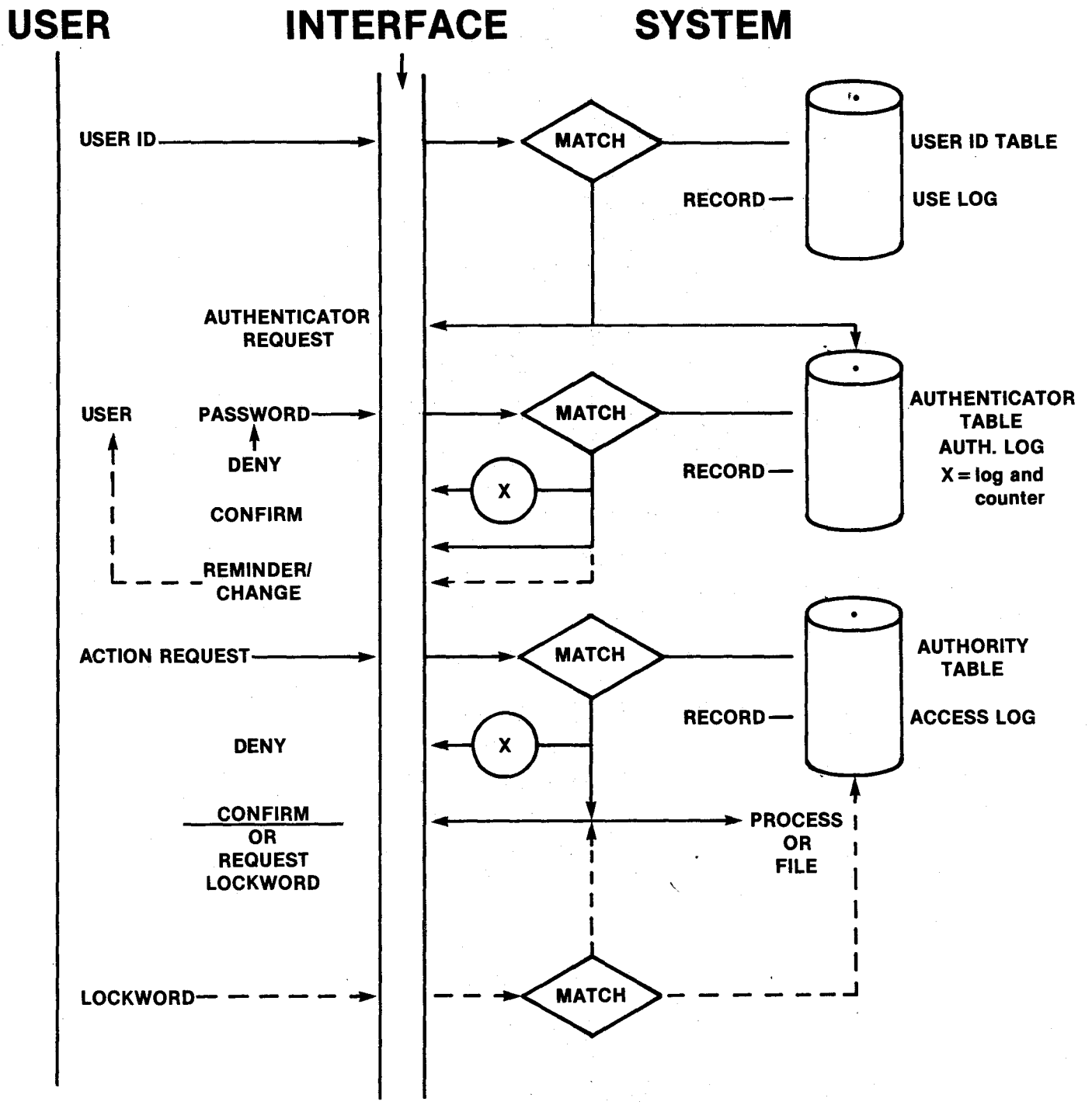
2. **Dial-up.** An indication of whether or not access via a dial-up port is allowed will be associated with each UserID. An attempt to use dial-up when not authorized will result in failure to sign-on.

   Anyone wanting to try and obtain a UserID/password combination by trial-and-error will probably need the privacy of a remote dial-up link to make the attempt. Therefore, limiting dial-up access to know UserID's who have valid reasons for dial-up, can limit this security risk.

3. **Unused UserID's.** If a UserID is unused for more than say 90 days, the computer logs that fact so that the System Security Administrator can ascertain if the UserID is still needed.

   A valuable aid for the System Security Administrator in isolating potentially defunct UserID's.

ACCESS MANAGEMENT



Computer Viruses

Theory and Experiments

Fred Cohen
University of Southern California
31 August 1984

1 Introduction and Abstract

This paper defines a major computer security problem called a virus. The virus is interesting because of its ability to attach itself to other programs and cause them to become viruses[1] as well. Given the wide spread use of sharing in current computer systems, the threat of a virus carrying a Trojan horse [Anderson 72] [Linde 75] is significant. Although a considerable amount of work has been done in implementing policies to protect from the illicit dissemination of information [Bell 73] [Denning 82], and many systems have been implemented to provide protection from this sort of attack [McCauley 79] [Popek 79] [Gold 79] [Landwehr 83], little work has been done in the area of keeping information entering an area from causing damage [Lampson 73] [Biba 77]. There are many types of information paths possible in systems, some legitimate and authorized, and others that may be covert [Lampson 73], the most commonly ignored one being through the user. We will ignore covert information paths throughout this paper.

The general facilities exist for providing provably correct protection schemes [Feiertag 79], but they depend on a security policy that is effective against the types of attacks being carried out. Even some quite simple protection systems cannot be proven 'safe' [Harrison 76]. Protection from denial of services requires the detection of halting programs which is well known to be undecidable [Garey 79]. The problem of precisely marking information flow within a system [Fenton 73] has been shown to be NP-complete. The use of guards for the passing of untrustworthy information [Woodward 79] between users has been examined, but in general depends on the ability to prove program correctness which is well known to be NP-complete.

The Xerox worm program [Shoch 82] has demonstrated the ability to propagate through a network, and has even accidentally caused denial of services. In a later variation, the game of 'core wars' [Dewdney 84] was invented to allow two programs to do battle with one another. Other variations on this theme have been reported by many unpublished authors, mostly in the context of night time games played between programmers. The term virus has also been used in conjunction with an augmentation to APL in which the author places a generic call at the beginning of each function which in turn invokes a preprocessor to augment the default APL interpreter [Gunn 74].

[1] There are two spellings for the plural of virus; 'virusses', and 'viruses'. We use the one found in Webster's 3rd International Unabridged Dictionary

The potential threat of a widespread security problem has been examined [Hoffman 82] and the potential damage to government, financial, business, and academic institutions is extreme. In addition, these institutions tend to use ad hoc protection mechanisms in response to specific threats rather than sound theoretical techniques [Kaplan 82]. Current military protection systems depend to a large degree on isolationism [Baker 83], however new systems are being developed to allow 'multilevel' usage [Klein 83]. None of the published proposed systems defines or implements a policy which could stop a virus.

In this paper, we open the new problem of protection from computer viruses. First we examine the infection property of a virus and show that the transitive closure of shared information could potentially become infected. When used in conjunction with a Trojan horse, it is clear that this could cause widespread denial of services and/or unauthorized manipulation of data. The results of several experiments with computer viruses are used to demonstrate that viruses are a formidable threat in both normal and high security operating systems. The paths of sharing, transitivity of information flow, and generality of information interpretation are identified as the key properties in the protection from computer viruses, and a case by case analysis of these properties is shown. Analysis shows that the only systems with potential for protection from a viral attack are systems with limited transitivity and limited sharing, systems with no sharing, and systems without general interpretation of information (Turing capability). Only the first case appears to be of practical interest to current society. In general, detection of a virus is shown to be undecidable both by a-priori and runtime analysis, and without detection, cure is likely to be difficult or impossible.

Several proposed countermeasures are examined and shown to correspond to special cases of the case by case analysis of viral properties. Limited transitivity systems are considered hopeful, but it is shown that precise implementation is intractable, and imprecise policies are shown in general to lead to less and less usable systems with time. The use of system wide viral antibodies is examined, and shown to depend in general on the solutions to intractable problems.

It is concluded that the the study of computer viruses is an important research area with potential applications to other fields, that current systems offer little or no protection from viral attack, and that the only provably 'safe' policy as of this time is isolationism.

2 A Computer Virus

We define a computer 'virus' as a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself. With the infection property, a virus can spread throughout a computer system or network using the authorizations of every user using it to infect their programs. Every program that gets infected may also act as a virus and thus the infection grows.

The following pseudo-program shows how a virus might be written in a pseudo-computer language. The ":=" symbol is used for definition, the ":" symbol labels a statement, the ";" separates statements, the "=" symbol is used for assignment or comparison, the "~" symbol stands for not, the "{" and "}" symbols group sequences of statements together, and the "..." symbol is used to indicate that an irrelevent portion of code has been left implicit.

```
program virus:=
{1234567;

subroutine infect-executable:=
        {loop:file = get-random-executable-file;
        if first-line-of-file = 1234567 then goto loop;
        prepend virus to file;
        }

subroutine do-damage:=
        {whatever damage is to be done}

subroutine trigger-pulled:=
        {return true if some condition holds}

main-program:=
        {infect-executable;
        if trigger-pulled then do-damage;
        goto next;}

next:}
```

A Simple Virus "V"

This example virus (V) searches for an uninfected executable file (E) by looking for executable files without the "1234567" in the beginning, and prepends V to E, turning it into an infected file (I). V then checks to see if some triggering condition is true, and does damage. Finally, V executes the rest of the program it was prepended to. When the user attempts to execute E, I is executed in its place; it infects another file and then executes as if it were E. With the exception of a slight delay for infection, I appears to be E until the triggering condition causes damage.

A common misconception of a virus relates it to programs that simply propagate through networks. The worm program, 'core wars', and other similar programs have done this, but none of them actually involve infection. The key property of a virus is its ability to infect other programs, thus reaching the transitive closure of sharing between users. As an example, if V infected one of user A's executables (E), and user B then ran E, V could spread to user B's files as well.

It should be pointed out that a virus need not be used for evil purposes or be a trojan horse. As an example, a compression virus could be written to find uninfected executables, compress them upon the user's permission, and prepend itself to them. Upon execution, the infected program decompresses itself and executes normally. Since it always asks permission before performing services, it is not a Trojan horse, but since it

has the infection property, it is still a virus. Studies indicate that such a virus could save over 50% of the space taken up by executable files in an average system. The performance of infected programs would decrease slightly as they are decompressed, and thus the compression virus implements a particular time space tradeoff. A sample compression virus could be written as follows:

```
program compression-virus:=
{01234567;

subroutine infect-executable:=
        {loop:file = get-random-executable-file;
        if first-line-of-file = 01234567 then goto loop;
        compress file;
        prepend compression-virus to file;
        }

main-program:=
        {if ask-permission then infect-executable;
        uncompress the-rest-of-this-file into tmpfile;
        run tmpfile;}
}
```

A Compression Virus 'C'

This program (C) finds an uninfected executable (E), compresses it, and prepends C to form an infected executable (I). It then uncompresses the rest of itself into a temporary file and executes normally. When I is run, it will seek out and compress another executable before decompressing E into a temporary file and executing it. The effect is to spread through the system compressing executable files, and decompress them as they are to be executed. Users will experience significant delays as their executables are decompressed before being run.

As a more threatening example, let us suppose that we modify the program V by specifying trigger-pulled as true after a given date and time, and specifying do-damage as an infinite loop. With the level of sharing in most modern systems, the entire system would likely become unusable as of the specified date and time. A great deal of work might be required to undo the damage of such a virus. This modification is shown here:

```
...
subroutine do-damage:=
        {loop: goto loop;}

subroutine trigger-pulled:=
        {if year>1984 then return true otherwise return false;}
...
```

A Denial of Services Virus

As an analogy to a computer virus, consider a biological disease that is 100% infectious, spreads whenever animals communicate, kills all infected animals instantly at a given moment, and has no detectable side effects until that moment. If a delay of even one week were used between the introduction of the disease and its effect, it would be very likely to leave only a few remote villages alive, and would certainly wipe out the

vast majority of modern society. If a computer virus of this type could spread throughout the computers of the world, it would likely stop most computer usage for a significant period of time, and wreak havoc on modern government, financial, business, and academic institutions.

3 Experiments with Computer Viruses

To demonstrate the feasibility of viral attack and the degree to which it is a threat, several experiments were performed. In each case, experiments were performed with the knowledge and consent of systems administrators. In the process of performing experiments, implementation flaws were meticulously avoided. It was critical that these experiments not be based on implementation lapses, but only on fundamental flaws in security policies.

The First Virus

On November 3, 1983, the first virus was conceived of as an experiment to be presented at a weekly seminar on computer security. The concept was first introduced in this seminar by the author, and the name 'virus' was thought of by Len Adleman. After 8 hours of expert work on a heavily loaded VAX 11/750 system running Unix, the first virus was completed and ready for demonstration. Within a week, permission was obtained to perform experiments, and 5 experiments were performed. On November 10, the virus was demonstrated to the security seminar.

The initial infection was implanted in 'vd', a program that displays Unix file structures graphically, and introduced to users via the system bulletin board. Since vd was a new program on the system, no performance characteristics or other details of its operation were known. The virus was implanted at the beginning of the program so that it was performed before any other processing.

In order to keep the attack under control several precautions were taken. All infections were performed manually by the attacker, and no damage was done, only reporting. Traces were included to assure that the virus would not spread without detection, access controls were used for the infection process, and the code required for the attack was kept in segments, each encrypted and protected to prevent illicit use.

In each of five attacks, all system rights were granted to the attacker in under an hour. The shortest time was under 5 minutes, and the average under 30 minutes. Even those who knew the attack was taking place were infected. In each case, files were 'disinfected' after experimentation to assure that no user's privacy would be violated. It was expected that the attack would be successful, but the very short takeover times were quite surprising. In addition, the virus was fast enough (under 1/2 second) that the delay to infected programs went unnoticed.

Once the results of the experiments were announced, administrators decided that no

further computer security experiments would be permitted on their system. This ban included the planned addition of traces which could track potential viruses and password augmentation experiments which could potentially have improved security to a great extent. This apparent fear reaction is typical, rather than try to solve technical problems technically, policy solutions are often chosen.

After successful experiments had been performed on a Unix system, it was quite apparent that the same techniques would work on many other systems. In particular, experiments were planned for a Tops-20 system, a VMS system, a VM/370 system, and a network containing several of these systems. In the process of negotiating with administrators, feasibility was demonstrated by developing and testing prototypes. Prototype attacks for the Tops-20 system were developed by an experienced Tops-20 user in 6 hours, a novice VM/370 user with the help of an experienced programmer in 30 hours, and a novice VMS user without assistance in 20 hours. These programs demonstrated the ability to find files to be infected, infect them, and cross user boundaries.

After several months of negotiation and administrative changes, it was decided that the experiments would not be permitted. The security officer at the facility was in constant opposition to security experiments, and would not even read any proposals. This is particularly interesting in light of the fact that it was offered to allow systems programmers and security officers to observe and oversee all aspects of all experiments. In addition, systems administrators were unwilling to allow sanitized versions of log tapes to be used to perform offline analysis of the potential threat of viruses, and were unwilling to have additional traces added to their systems by their programmers to help detect viral attacks. Although there is no apparent threat posed by these activities, and they require little time, money, and effort, administrators were unwilling to allow investigations. It appears that their reaction was the same as the fear reaction of the Unix administrators.

A Bell-LaPadula Based System

In March of 1984, negotiations began over the performance of experiments on a Bell-LaPadula [Bell 73] based system implemented on a Univac 1108. The experiment was agreed upon in principal in a matter of hours, but took several months to become solidified. In July of 1984, a two week period was arranged for experimentation. The purpose of this experiment was merely to demonstrate the feasibility of a virus on a Bell-LaPadula based system by implementing a prototype.

Because of the extremely limited time allowed for development (26 hours of computer usage by a user who had never used an 1108, with the assistance of a programmer who hadn't used an 1108 in 5 years), many issues were ignored in the implementation. In particular, performance and generality of the attack were completely ignored. As a result, each infection took about 20 seconds, even though they could easily have been done in under a second. Traces of the virus were left on the system although they could

have been eliminated to a large degree with little effort. Rather than infecting many files at once, only one file at a time was infected. This allowed the progress of a virus to be demonstrated very clearly without involving a large number of users or programs. As a security precaution, the system was used in a dedicated mode with only a system disk, one terminal, one printer, and accounts dedicated to the experiment.

After 18 hours of connect time, the 1108 virus performed its first infection. The host provided a fairly complete set of user manuals, use of the system, and the assistance of a competent past user of the system. After 26 hours of use, the virus was demonstrated to a group of about 10 people including administrators, programmers, and security officers. The virus demonstrated the ability to cross user boundaries and move from a given security level to a higher security level. Again it should be emphasized that no system bugs were involved in this activity, but rather that the Bell-LaPadula model allows this sort of activity to legitimately take place.

All in all, the attack was not difficult to perform. The code for the virus consisted of 5 lines of assembly code, about 200 lines of Fortran code, and about 50 lines of command files. It is estimated that a competent systems programmer could write a much better virus for this system in under 2 weeks. In addition, once the nature of a viral attack is understood, developing a specific attack is not difficult. Each of the programmers present was convinced that they could have built a better virus in the same amount of time. (This is believable since this attacker had no previous 1108 experience.)

Instrumentation

In early August of 1984, permission was granted to instrument a VAX Unix system to measure sharing and analyze viral spreading. Data at this time is quite limited, but several trends have appeared. The degree of sharing appears to vary greatly between systems, and many systems may have to be instrumented before these deviations are well understood. A small number of users appear to account for the vast majority of sharing, and a virus could be greatly slowed by protecting them. The protection of a few 'social' individuals might also slow biological diseases. The instrumentation was conservative in the sense that infection could happen without the instrumentation picking it up, so estimated attack times are unrealistically slow.

As a result of the instrumentation of these systems, a set of 'social' users were identified. Several of these surprised the main systems administrator. The number of systems administrators was quite high, and if any of them were infected, the entire system would likely fall within an hour. Some simple procedural changes were suggested to slow this attack by several orders of magnitude without reducing functionality.

Summary of Spreading

system 1				system 2			
class	##	spread	time	class	##	spread	time
S	3	22	0	S	5	160	1
A	1	1	0	A	7	78	120
U	4	5	18	U	7	24	600

Two systems are shown, with three classes of users (S for system, A for system administrator, and U for normal user). '##' indicates the number of users in each catagory, 'spread' is the average number of users a virus would spread to, and 'time' is the average time taken to spread them once they logged in, rounded up to the nearest minute. Average times are misleading because once an infection has reaches the 'root' account on Unix, all access is granted. Taking this into account leads to takeover times on the order of one minute which is so fast that infection time becomes a limiting factor in how quickly infections can spread. This coincides with previous experimental results using an actual virus.

Users who were not shared with are ignored in these calculations, but other experiments indicate that any user can get shared with by offering a program on the system bulletin board. Detailed analysis demonstrated that systems administrators tend to try these programs as soon as they are announced. This allows normal users to infect system files within minutes. Administrators used their accounts for running other users' programs and storing commonly executed system files, and several normal users owned very commonly used files. These conditions make viral attack very quick. The use of seperate accounts for systems administrators during normal use was immediately suggested, and the systematic movement (after verification) of commonly used programs into the system domain was also considered.

Summary and Conclusions

The following table summarizes the results of the experiments to date. The three systems are across the horizontal axis (Unix, Bell-LaPadula, and Instrumentation), while the vertical axis indicates the measure of performance (time to program, infection time, number of lines of code, number of experiments performed, minimum time to takeover, average time to takeover, and maximum time to takeover) where time to takeover indicates that all privilages would be granted to the attacker within that delay from introducing the virus.

Summary of Attacks

	Unix	B-L	Instr
Time	8 hrs	18 hrs	N/A
Inf t	.5 sec	20 sec	N/A
Code	200 l	260 l	N/A
Trials	5	N/A	N/A
Min t	5 min	N/A	30 sec
Avg t	30 min	N/A	30 min
Max t	60 min	N/A	48 hrs

Viral attacks appear to be easy to develop in a very short time, can be designed to leave few if any traces in most current systems, are effective against modern security policies for multilevel usage, and require only minimal expertise to implement. Their potential threat is severe, and they can spread very quickly through a computer system. It appears that they can spread through computer networks in the same way as they spread through computers, and thus present a widespread and fairly immediate threat to many current systems.

The problems with policies that prevent controlled security experiments are clear; denying users the ability to continue their work promotes illicit attacks; and if one user can launch an attack without using system bugs or special knowledge, other users will also be able to. By simply telling users not to launch attacks, little is accomplished; users who can be trusted will not launch attacks; but users who would do damage cannot be trusted, so only legitimate work is blocked. The perspective that every attack allowed to take place reduces security is in the author's opinion a fallacy. The idea of using attacks to learn of problems is even required by government policies for trusted systems [Klein 83] [Kaplan 82]. It would be more rational to use open and controlled experiments as a resource to improve security.

4 Prevention of Computer Viruses

We have introduced the concept of viruses to the reader, and actual viruses to systems. Having planted the seeds of a potentially devastating attack, it is appropriate to examine protection mechanisms which might help defend against it. We examine here prevention of computer viruses.

Basic Limitations

In order for users of a system to be able to share information, there must be a path through which information can flow from one user to another. Given a general purpose

system in which users are capable of using information in their possession as they wish, it should be clear that the ability to share information is transitive. That is, if there is a path from user A to user B, and there is a path from user B to user C, then there is a path from user A to user C with the witting or unwitting cooperation of user B. Finally, there is no fundamental distinction between information that can be used as data, and information that can be used as program. This can be clearly seen in the case of an interpreter that takes information edited as data, and interprets it as a program. In effect, information only has meaning in that it is subject to interpretation. Sharing, transitivity of information flow, and generality of interpretation allow a virus to spread to the transitive closure of information flow starting at any given source.

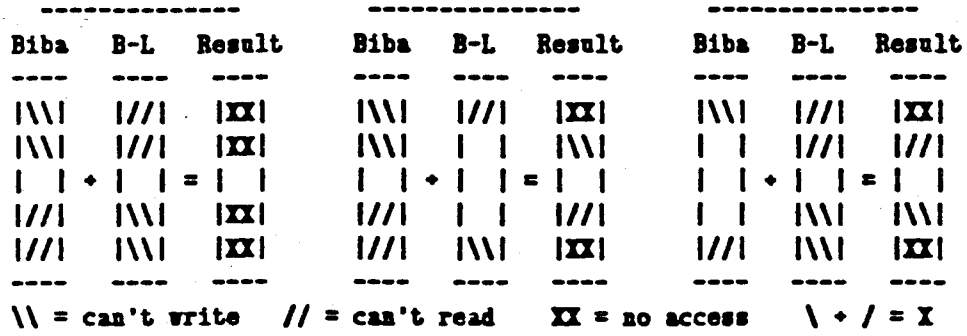
Clearly, if there is no sharing, there can be no infection or illicit dissemination of information. This is called 'isolationism'. Just as clearly, a system in which no program can be altered, and information cannot be used to make decisions, cannot be infected. We call this a 'fixed first order functionality' system. We should note that virtually any system with real usefulness in a scientific or development environment will require generality of interpretation, and that isolationism is unacceptable if we wish to benefit from the work of others. Nevertheless, these are solutions to the problem of viruses which may be applicable in limited situations.

Two limits on the paths of information flow can be distinguished, those that partition users into closed proper subsets under transitivity, and those that don't. Flow restrictions that result in closed subsets can be viewed as partitions of a system into isolated subsystems. These limit each infection to one partition. This is a viable means of preventing complete viral takeover at the expense of limited isolationism. It is equivalent to giving each isolated subset of users their own computer, and can be simplified to independent analysis of each subsystem.

The Integrity Model

The integrity model [Biba 77] is an example of a policy that partitions systems into closes subsets under transitivity. In the Biba model, an integrity level is associated with all information. The strict integrity properties are the dual of the Bell-LaPadula properties; no user at a given integrity level can read an object of lower integrity or write an object of higher integrity. In Biba's original model, a distinction was made between read and execute access, but this cannot be enforced without restricting the generality of information interpretation. Since a high integrity program can write a low integrity object, it can make low integrity copies of itself, and then read low integrity input and produce low integrity output.

If the integrity model and the Bell-LaPadula model coexist, a form of limited isolationism results which divides the space into closed subsets under transitivity. If the same divisions are used for both mechanisms (higher integrity corresponds to higher security), isolationism results since information moving up security levels also moves up integrity levels, and this is not permitted. This is shown graphically below:



Biba's work also included two other integrity policies, the 'low water mark' policy which makes output the lowest integrity of any input, and the 'ring' policy in which users cannot invoke everything they can read. The former policy tends to move all information towards lower integrity levels, while the latter attempts to make a distinction that cannot be made with generalized information interpretation.

Just as systems based on the Bell-LaPadula model tend to cause all information to move towards higher levels of security by always increasing the level to meet the highest level user, the Biba model tends to move all information towards lower integrity levels by always reducing the integrity of results to that of the lowest incoming integrity. We also know that a precise system for integrity is NP-complete (just as its dual is NP-complete).

The most trusted programmer is (by definition) the programmer that can write programs executable by the most users. In order to maintain the the Bell-LaPadula policy, high level users cannot write programs used by lower level users. This means that the most trusted programmers must be those at the lowest security level. Thus the highest integrity programs must come from the lowest security level users. This seems contradictory.

Flow Models

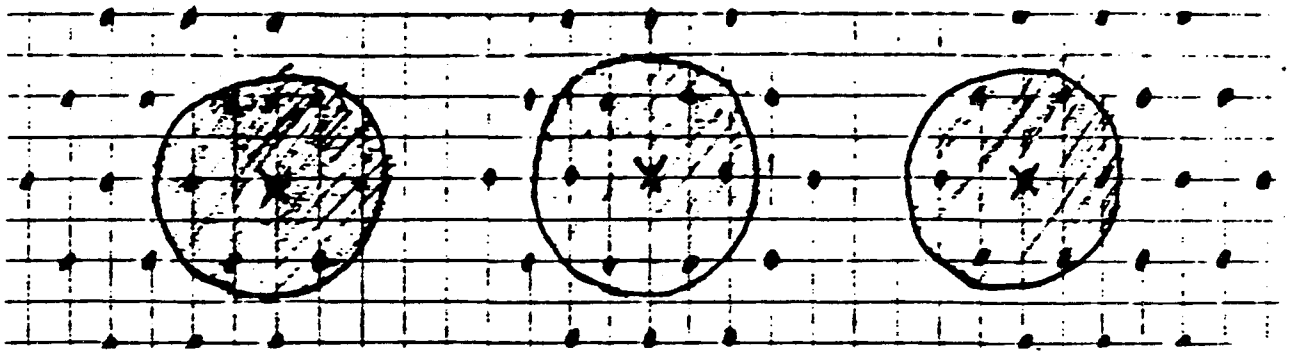
In policies that don't partition systems into closed proper subsets under transitivity, it is possible to limit the extent over which a virus can spread. The 'flow distance' policy implements a distance metric by keeping track of the distance (number of sharings) over which data has flowed. The rules are; the distance of output information is the maximum of the distances of input information, and the distance of shared information is one more than the distance of the same information before sharing. Protection is provided by enforcing a threshold above which information becomes unusable. Thus a file with distance 8 shared into a process with distance 2 increases the process to distance 9, and any further output will be at at least that distance.

As an example, we show the access allowed to three different users in a distance metric system with the threshold set at 1 and each user (represented by a dot) able to communicate with only the 6 nearest neighbors. Notice that each user can access their

neighbors information, but that any information their neighbor has that is shared from a user at a distance of more than 1 is not accessible.

Rules:

- D(output) = max(D(input))
- D(shared input)=1+D(unshared input)
- Information is accessible iff D < const



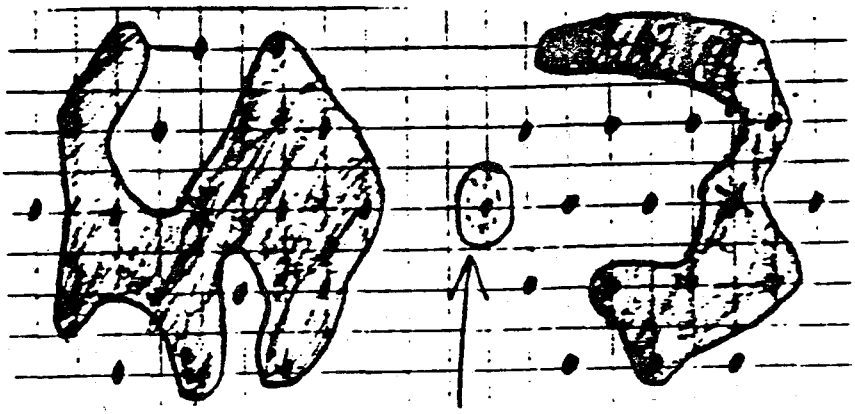
A Distance Metric with a Threshold of 1

The 'flow list' policy maintains a list of all users who have had an effect on each object. The rule for maintaining this list is; the flow list of output is the union of the flow lists of all inputs (including the user who causes the action). Protection takes the form of an arbitrary boolean expression on flow lists which determines accessibility. This is a very general policy, and can be used to represent any of the above policies by selecting proper boolean expressions. A mechanism has been designed for flow lists under Unix, but has not been implemented as of this writing.

The following figure shows an example of a flow list system implementing different restrictions for different users. Notice that there is a user that could be accessed if there were a path from that user to the second user, but that since there is no path, no sharing can actually take place. In addition, as in the distance metric system, information accessed by the first user cannot necessarily be accessed by the second user even though the second user can access information of the first user not shared from other users.

Rules:

- F(output)=Union(F(inputs))
- Information is accessible iff B(F)=1



A Sample Flow List System

In a system with unlimited information paths, limited transitivity may have an effect if users don't use all available paths, but since there is always a direct path between any two users, there is always the possibility of infection. As an example, in a system with transitivity limited to a distance of 1 it is 'safe' to share information with any user you 'trust' without having to worry about whether that user has incorrectly trusted another user.

Limited Interpretation

With limits on the generality of interpretation less restrictive than fixed first order interpretation, the ability to infect is an open question because infection depends on the functions permitted. Certain functions are required for infection. The ability to write is required, but any useful program must have output. It is possible to design a set of operations that don't allow infection in even the most general case of sharing and transitivity, but it is not known whether any such set includes non fixed first order functions. Although no fixed interpretation scheme can itself be infected, a high order fixed interpretation scheme can be used to infect programs written to be interpreted by it. As an example, the microcode of a computer may be fixed, but the machine language it interprets can still be infected. Systems with limited interpretation will not be able to spread infections any further than those with general interpretation, so the previous results provide upper bounds on the spread of a virus in systems with limited sharing.

Precision Problems

Although isolationism and limited transitivity offer solutions to the infection problem, they are not ideal in the sense that widespread sharing is generally considered a valuable tool in computing. Of these policies, only isolationism can be precisely implemented in practice because tracing exact information flow requires NP-complete time, and

maintaining markings requires large amounts of space [Denning 82]. This leaves us with imprecise techniques. The problem with imprecise techniques is that they tend to move systems towards isolationism. This is because they use conservative estimates of effects in order to prevent potential damage. The philosophy behind this is that it is better to be safe than sorry.

The problem is that when information has been unjustly deemed unreadable by a given user, the system becomes less usable for that user. This is a form of denial of services in that access to information that should be accessible is denied. Such a system always tends to make itself less and less usable for sharing until it either becomes completely isolationist or reaches a stability point where all estimates are precise. If such a stability point existed, we would have a precise system for that stability point. Since we know that any precise stability point besides isolationism requires the solution to an NP-complete problem, we know that any non NP-complete solution must tend towards isolationism.

Summary and Conclusions

The following table summarizes the preventative protection from computer viruses just examined.

	Limits of viral infection			
	general interpretation		limited interpretation	
\Transitivity Sharing\	limited	general	limited	general
general	unlimited	unlimited	unknown	unknown
limited	arbitrary	closure	arbitrary	closure

5 Cure of Computer Viruses

Since prevention of computer viruses may be infeasible if widespread sharing is desired, the biological analogy leads us to the possibility of cure as a means of protection. Cure in biological systems depends on the ability to detect a virus and find a way to overcome it. A similar possibility exists for computer viruses. We now examine the potential for detection and removal of a computer virus.

Detection of Viruses

In order to determine that a given program 'P' is a virus, it must be determined that P infects other programs. This is undecidable since P could invoke the decision procedure 'D' and infect other programs if and only if D determines that P is not a virus. We conclude that a program that precisely discerns a virus from any other program by examining its appearance is infeasible. In the following modification to program V, we

use the hypothetical decision procedure D which returns "true" iff its argument is a virus, to exemplify the undecidability of D.

```
program contradictory-virus:=
{...
main-program:=
        {if ~D(contradictory-virus) then
                {infect-executable;
                if trigger-pulled then do-damage;
                }
        goto next;
        }
}
```

Contradiction of the Decidability of a Virus "CV"

By modifying the main-program of V, we have assured that if the decision procedure D determines CV to be a virus, CV will not infect other programs, and thus will not act as a virus. Contrapositively, if D determines that CV is not a virus, CV will infect other programs, and thus be a virus. Therefore, the hypothetical decision procedure D is made self contradictory, and precise determination of a virus by its appearance is undecidable.

Evolutions of a Virus

In our experiments, the virus took less than 4000 bytes to implement on a general purpose computer. Since we could interleave ány program that doesn't halt, terminates in finite time, and doesn't overwrite the virus or any of its state variables, and still have a virus, the number of possible variations on a single virus is clearly very large. In this example of an evolutionary virus EV, we augment V by allowing it to add random statements between any two necessary statements.

```
program evolutionary-virus:=
{...
subroutine print-random-statement:=
        {print random-variable-name, " = ", random-variable-name;
        loop:if random-bit = 0 then
                {print random-operator, random-variable-name;
                goto loop;}
        print semicolon;
        }

subroutine copy-virus-with-random-insertions:=
        {loop: copy evolutionary-virus to virus till semicolon-found;
        if random-bit = 1 then print-random-statement;
        if ^end-of-input-file goto loop;
        }

main-program:=
        {copy-virus-with-random-insertions;
        infect-executable;
        if trigger-pulled do-damage;
        goto next;}

next:}
```

An Evolutionary Virus "EV"

In general, proof of the equivalence of two evolutions of a program 'P' ('P1' and 'P2') is undecidable because any decision procedure 'D' capable of finding their equivalence could be invoked by P1 and P2. If found equivalent they perform different operations, and if found different they act the same, and are thus equivalent. This is exemplified by the following modification to program EV in which the decision procedure D returns "true" iff two input programs are equivalent.

```
program undecidable-evolutionary-virus:=
{...
subroutine copy-with-undecidable-assertion:=
        {copy undecidable-evolutionary-virus to file till line-starts-with-zzz;
        if file = P1 then print "if D(P1,P2) then print 1;";
        if file = P2 then print "if D(P1,P2) then print 0;";
        copy undecidable-evolutionary-virus to file till end-of-input-file;
        }

main-program:=
        {if random-bit = 0 then file = P1 otherwise file = P2;
        copy-with-undecidable-assertion;
        zzz:
        infect-executable;
        if trigger-pulled do-damage;
        goto next;}

next:}
```

Undecidable Equivalence of Evolutions of a Virus "UEV"

The program UEV evolves into one of two types of programs P1 or P2. If the program type is P1, the statement labeled "zzz" will become:

if D(P1,P2) then print 1;

while if the program type is P2, the statement labeled "zzz" will become:

if D(P1,P2) then print 0;

The two evolutions each call decision procedure D to decide whether they are equivalent. If D indicates that they are equivalent, then P1 will print a 1 while P2 will print a 0, and D will be contradicted. If D indicates that they are different, neither prints anything. Since they are otherwise equal, D is again contradicted. Therefore, the hypothetical decision procedure D is self contradictory, and the precise determination of the equivalence of these two programs by their appearance is undecidable.

Since both P1 and P2 are evolutions of the same program, the equivalence of evolutions of a program is undecidable, and since they are both viruses, the equivalence of evolutions of a virus is undecidable. Program UEV also demonstrates that two unequivalent evolutions can both be viruses. The evolutions are equivalent in terms of their viral effects, but may have slightly different side effects.

An alternative to detection by appearance, is detection by behavior. A virus, just as any other program, acts as a surrogate for the user in requesting services, and the services used by a virus are legitimate in legitimate uses. The behavioral detection question then becomes one of defining what is and is not a legitimate use of a system service, and finding a means of detecting the difference.

As an example of a legitimate virus, a compiler that compiles a new version of itself is in fact a virus by the definition given here. It is a program that 'infects' another program by modifying it to include an evolved version of itself. Since the viral capability is in most compilers, every use of a compiler is a potential viral attack. The viral activity of a compiler is only triggered by particular inputs, and thus in order to detect triggering, one must be able to detect a virus by its appearance. Since precise detection by behavior in this case depends on precise detection by the appearance of the inputs, and since we have already shown that precise detection by appearance is undecidable, it follows that precise detection by behavior is also undecidable.

Limited Viral Protection

A limited form of virus has been designed [Thompson 84] in the form of a special version of the C compiler that can detect the compilation of the login program and add a Trojan horse that lets the author login. Thus the author could access any Unix system with this compiler. In addition, the compiler can detect compilations of new versions of itself and infect them with the same Trojan horse. Whether or not this has actually been implemented is unknown (although many say the NSA has a working version of it).

As a countermeasure, we can devise a new login program (and C compiler) sufficiently different from the original as to make its equivalence very difficult to determine. If the 'best AI program of the day' would be incapable of detecting their equivalence in a given amount of time, and the compiler performed its task in less than that much time, it

could be reasonably assumed that the virus could not have detected the equivalence, and therefore would not have propagated itself. If the exact nature of the detection were known, it would likely be quite simple to work around it.

Although we have shown that in general it is impossible to detect viruses, any particular virus can be detected by a particular detection scheme. For example, virus V could easily be detected by looking for 1234567 as the first line of an executable. If the executable were found to be infected, it would not be run, and would therefore not be able to spread. The following program is used in place of the normal run command, and refuses to execute programs infected by virus V:

```
program new-run-command:=
{file = name-of-program-to-be-executed;
if first-line-of-file = 1234567 then
        {print "the program has a virus";
        exit;}
otherwise run file;
}
```

Protection from Virus V 'PV'

Similarly, any particular detection scheme can circumvented by a particular virus. As an example, if an attacker knew that a user was using the program PV as protection from viral attack, the virus V could easily be substituted with a virus V' where the first line was 123456 instead of 1234567. Much more complex defense schemes and viruses can be examined. What becomes quite evident is analogous to the old western saying: "ain't a horse that can't be rode, ain't a man that can't be throwed". No infection can exist that cannot be detected, and no detection mechanism can exist that can't be infected.

This result leads to the idea that a balance of coexistent viruses and defenses could exist, such that a given virus could only do damage to a given portion of the system, while a given protection scheme could only protect against a given set of viruses. If each user and attacker used identical defenses and viruses, there could be an ultimate virus or defense. It makes sense from both the attacker's point of view and the defender's point of view to have a set of (perhaps incompatible) viruses and defenses.

In the case where viruses and protection schemes didn't evolve, this would likely lead to some set of fixed survivors, but since programs can be written to evolve, the program that evolved into a difficult to attack program would more likely survive as would a virus that was more difficult to detect. As evolution takes place, balances tend to change, with the eventual result being unclear in all but the simplest circumstances. This has very strong analogies to biological theories of evolution [Dawkins 78], and might relate well to genetic theories of diseases. Similarly, the spread of viruses through systems might well be analyzed by using mathematical models used in the study of infectious diseases [Baily 57].

Since we cannot precisely detect a virus, we are left with the problem of defining potentially illegitimate use in a decidable and easily computable way. We might be

willing to detect many programs that are not viruses and even not detect some viruses in order to detect a large number of viruses. If an event is relatively rare in 'normal' use, it has high information content when it occurs, and we can define a threshold at which reporting is done. If sufficient instrumentation is available, flow lists can be kept which track all users who have effected any given file. Users that appear in many incoming flow lists could be considered suspicious. The rate at which users enter incoming flow lists might also be a good indicator of a virus.

This type of measure could be of value if the services used by viruses are rarely used by other programs, but presents several problems. If the threshold is known to the attacker, the virus can be made to work within it. An intelligent thresholding scheme could adapt so the threshold could not be easily determined by the attacker. Although this 'game' can clearly be played back and forth, the frequency of dangerous requests might be kept low enough to slow the undetected virus without interfering significantly with legitimate use.

Several systems were examined for their abilities to detect viral attacks. Surprisingly, none of these systems even include traces of the owner of a program run by other users. Marking of this sort must almost certainly be used if even the simplest of viral attacks are to be detected.

Once a virus is implanted, it may not be easy to fully remove. If the system is kept running during removal, a disinfected program could be reinfected. This presents the potential for infinite tail chasing. Without some denial of services, removal is likely to be impossible unless the program performing removal is faster at spreading than the virus being removed. Even in cases where the removal is slower than the virus, it may be possible to allow most activities to continue during removal without having the removal process be very fast. For example, one could isolate a user or subset of users and cure them without denying services to other users.

In general, precise removal depends on precise detection, because without precise detection, it is impossible to know precisely whether or not to remove a given object. In special cases, it may be possible to perform removal with an inexact algorithm. As an example, every file written after a given date could be removed in order to remove any virus started after that date.

Spontaneous Generation of a Virus

One concern that can be easily laid to rest is the chance that a virus could be spontaneously generated from some error in the hardware or on the part of a programmer. As a simple example, let us suppose that we start with a memory containing no error detection or correction, with a length of only 1000 bits, prepared ahead of time with a program that requires only a single bit change to turn it into a virus. If random bit changes are allowed, then the chances of the desired bit changing is about 1 in 1000.

If we assume that one of the other 999 bits changed instead, the chances of the program becoming the virus are now significantly reduced since two events must both happen. The chances of the first are 2 in 1000, and the chances of the second are 1 in 1000, so the resulting probability is 2 in 1,000,000. It is clear that the chances of spontaneous generation from a program that is very nearly a virus decrease with time, and thus the major threat of such a mutation is very short run. This may relate to the reason that a mutant fetus will often die early in pregnancy, but once it has survived for several months, its chances of birth are significantly increased.

If we start with a randomly prepared memory, the chances are that 50% of the bits would have to change for it to become a specific virus, and the probability of spontaneous generation becomes astronomically small. Since 500 bit changes would be required in the example case to mutate into a virus from this condition, the chances are on the order of $500!/1000^{500}$ which is indeed astronomically small.

We now note that the simple virus is less likely to succeed in the long run than the more advanced viruses because it is not so sophisticated that it cannot be easily detected with simple techniques. In order to write evolutionary viruses, considerably more effort was required, and a correspondingly longer virus resulted. In order to have these viruses general purpose enough to infect many types of programs, they would likely have to be still more complex. Since the probability of spontaneous generation goes down so quickly with increasing lengths, the chances of a spontaneously generated difficult to detect virus is still smaller.

If we examine most modern computers, the probability of a random set of bit errors going undetected is very small. In many systems, the chances are less than 1 such error per million hours of operation with a memory size of many million bytes. In order to have a spontaneous generation of this sort, we would have to have the correct set of bit errors happen. The number of possible resulting combinations is approximately exponential in the length of the desired result, and the chances of any given set of bits resulting from such a bit error are inversely proportional to the number of possible results. We conclude that the chances of spontaneous generation from random errors is ignorable.

Perhaps a much more realistic concern is that a programmer will accidentally create a virus in the process of trying to create an evolutionary program or a compiler. This too is probably unlikely in comparison to the chances that an attacker would try to intentionally create a virus.

6 Summary, Conclusions, and Further Work

To quickly summarize, absolute protection can be easily attained by absolute isolationism, but that is usually an unacceptable solution. Other forms of protection all seem to depend on the use of extremely complex and/or resource intensive analytical techniques, or imprecise solutions that tend to make systems less usable with time.

Prevention appears to involve restricting legitimate activities, while cure may be arbitrarily difficult without some denial of services. Precise detection is undecidable, however statistical methods may be used to limit undetected spreading either in time or in extent. Behavior of typical usage must be well understood in order to use statistical methods, and this behavior is liable to vary from system to system. Limited forms of detection and prevention could be used in order to offer limited protection from viruses.

It has been demonstrated that a virus has the potential to spread throughout any system which allows sharing. Every general purpose system currently in use is open to viral attack. In current 'secure' systems, viruses tend to spread further when created by less trusted users. Experiments show the viability of viral attack, and indicate that viruses spread quickly and are easily created on a variety of operating systems. Further experimentation is still underway.

The results presented are not operating system or implementation specific, but are based on the fundamental properties of systems. More importantly, they reflect realistic assumptions about systems currently in use. Further, nearly every 'secure' system currently under development is based on the Bell-LaPadula or lattice policy alone, and this work has clearly demonstrated that these models are insufficient to prevent viral attack.

Several undecidable problems have been identified with respect to viruses and countermeasures. The are summarized here:

Undecidable Detection Problems

Detection of a virus by its appearance
Detection of a virus by its behavior
Detection of an evolution of a known virus
Detection of a triggering mechanism by its appearance
Detection of a triggering mechanism by its behavior
Detection of an evolution of a known triggering mechanism
Detection of a virus detector by its appearance
Detection of a viral detector by its behavior
Detection of an evolution of a known viral detector

Several potential countermeasures were examined in some depth, and none appear to offer ideal solutions. Several of the techniques suggested in this paper which could offer limited viral protection are designed and being considered for implementation. To be perfectly secure against viral attacks, a system must protect against incoming information flow, while to be secure against leakage of information a system must protect against outgoing information flow. In order for systems to allow sharing, there must be some information flow. It is therefore the major conclusion of this paper that the goals of sharing in a general purpose multilevel security system may be in such direct opposition to the goals of viral security as to make their reconciliation and coexistence impossible.

The most important ongoing research involves determining how quickly a virus could spread to a large percentage of the computers in the world. This is being done through simplified mathematical models and studies of viral spreading in 'typical' computer networks. Significant examples of evolutionary programs have been developed at the source level for producing many evolutions of a given program. A simple evolving virus has been developed, and a simple evolving antibody is also under development. A genetic theory of computer viruses is under consideration, but no solid progress has been made in this area. A flow list mechanism for Unix will be implemented when the necessary hardware is available, and the instrumentation of networks is expected to continue as long as facilities and funding permit. Statistical detection techniques based on the results of instrumentation are also in the planning stages.

7 Acknowledgements

Because of the sensitive nature of much of this research and the experiments performed in its course, many of the people to whom I am greatly indebted cannot be explicitly thanked. Rather than ignoring anyone's help, I have decided to give only first names. Len and David have provided a lot of good advice in both the research and writing of this paper, and without them I likely would never have gotten it to this point. John, Frank, Connie, Chris, Peter, Terry, Dick, Jerome, Mike, Marv, Steve, Lou, Steve, Andy, and Loraine all put their noses on the line more than just a little bit in their efforts to help perform experiments, publicize results, and lend covert support to the work. Martin, John, Magdy, Xi-an, Satish, Chris, Steve, JR, Jay, Bill, Fadi, Irv, Saul, and Frank all listened and suggested, and their patience and friendship were invaluable. Alice, John, Mel, Ann, and Ed provided better blocking than the USC front 4 ever has.

References

[Anderson 72] J. P. Anderson. *Computer Security Technology Planning Study.* Technical Report ESD-TR-73-51, USAF Electronic Systems Division, Oct, 1972. Cited in Denning.

[Baily 57] Norman T. J. Baily. *The Mathematical Theory of Epidemics.* Hafner Publishing Co., N.Y., 1957.

[Baker 83] D. B. Baker. *Department of Defense Trusted Computer System Evaluation Criteria (Final Draft).* private communication, The Aerospace Corporation, 1983.

[Bell 73] D. E. Bell and L. J. LaPadula. *Secure Computer Systems: Mathematical Foundations and Model.* The Mitre Corporation, 1973. cited in many papers.

[Biba 77] K. J. Biba. *Integrity Considerations for Secure Computer Systems.* USAF Electronic Systems Division, 1977. cited in Denning.

[Dawkins 78] Richard Dawkins. *The Selfish Gene.* Oxford Press, N.Y., N.Y., 1978.

[Denning 82] D. E. Denning. *Cryptography and Data Security.* Addison Wesley, 1982.

[Dewdney 84] A.K.Dewdney. Computer Recreations. *Scientific American* 250(5):14-22, May, 84.

[Feiertag 79] R. J. Feiertag and P.G. Neumann. The foundations of a Provable Secure Operating System (PSOS). In *National Computer Conference*, pages 329-334. AIFIPS, 1979.

[Fenton 73] J. S. Fenton. *Information Protection Systems.* PhD thesis, U. of Cambridge, 1973. Cited in Denning.

[Garey 79] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, 1979.

[Gold 79] B. D. Gold, R. R. Linde, R. J. Peeler, M. Schaefer, J.F. Scheid, and P.D. Ward. A Security Retrofit of VM/370. In *National Computer Conference*, pages 335-344. AIFIPS, 1979.

(Gunn 74) ACM. *Use of Virus Functions to Provide a Virtual APL Interpreter Under User Control*, 1974.

[Harrison 76] M. A. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in Operating Systems. In *Proceedings.* ACM, 1976.

[Hoffman 82] L. J. Hoffman. Impacts of information system vulnerabilities on society. In *National Computer Conference*, pages 461-467. AIFIPS, 1982.

[Kaplan 82] U.S. Dept. of Justice, Bureau of Justice Statistics. *Computer Crime - Computer Security Techniques*. U.S. Government Printing Office, Washington, DC, 1982.

[Klein 83] M. H. Klein. *Department of Defense Trusted Computer System Evaluation Criteria*. Department of Defense, Fort Meade, Md. 20755, 1983.

[Lampson 73] B. W. Lampson. A note on the Confinement Problem. In *Communications*. ACM, Oct, 1973.

[Landwehr 83] C. E. Landwehr. The Best Available Technologies for Computer Security. *Computer* 16(7), July, 1983.

[Linde 75] R. R. Linde. Operating System Penetration. In *National Computer Conference*, pages 361-368. AIFIPS, 1975.

[McCauley 79] E. J. McCauley and P. J. Drongowski. KSOS - The Design of a Secure Operating System. In *National Computer Conference*, pages 345-353. AIFIPS, 1979.

[Popek 79] G.J. Popek, M. Kampe, C.S. Kline, A. Stoughton, M. Urban, and E.J. Walton. UCLA Secure Unix. In *National Computer Conference*. AIFIPS, 1979.

(Shoch 82) ACM. *The 'Worm' Programs - Early Experience with a Distributed Computation*, 1982.

(Thompson 84) ACM. *Reflections on Trusting Trust*, 1984.

[Woodward 79] J. P. L. Woodward. Applications for Multilevel Secure Operating Systems. In *National Computer Conference*, pages 319-328. AIFIPS, 1979.

PASSWORD MANAGEMENT

IN PRACTICE

Sheila L. Brand
and
Mary E. Flaherty

DoD Computer Security Center

1.0 Introduction

The requirement to protect data from being accessed by unauthorized users has been evident for quite some time. Recent publicity describing situations where unauthorized accesses have occurred has made large numbers of people aware of the problem. The relative ease with which these accesses have been accomplished has emphasized the need to implement procedures which will minimize, if not eliminate, these occurrences.

Key to preventing unauthorized access is the ability to provide reliable and relatively uncircumventable user identification and authentication mechanisms in the computer system charged with protection of that data. Several techniques have been advanced as "fool proof" authentication methods (e.g. signature, handprint, fingerprint verification, etc). However, to date, the most common system in use is that of the password. Therefore, implementation of a secure password management system would go far towards raising the deterrent level against unauthorized access attempts.

The DoD Computer Security Center, recognizing the need for guidance in this area, has developed a guideline to address issues related to password management. Salient features of this guideline, titled: A Guideline on Password Management, are described in this paper.

2.0 Scope

Specific areas addressed in this guideline include the responsibilities of the System Security Officer and of users, functionality of the authentication mechanism, and password generation. Major features advocated in this guideline are:

* Users should be able to change their own passwords

* Passwords should be machine generated rather than user-supplied

* Audit reports such as date and time of last login should be provided by the system directly to the user

3.0 Control Objectives

These advocated features and other prescribed safeguards are derived from the following Control Objectives for password systems.

3.1 Personal Identification

Password systems used to control access to ADP systems that process or handle classified or other sensitive information must assure the capability to uniquely identify each individual user of the system.

3.2 Authentication

Password systems used to control access to ADP systems that process or handle classified or other sensitive information must result in unequivocal authentication of the user's claimed identity.

3.3 Password Privacy

Password systems must assure, to the extent possible, compromise protection of the password database consistent with protection afforded the classified or other sensitive information processed or handled by the ADP system in which the password systems operate.

3.4 Auditing

Password systems used to control access to ADP systems that process or handle classified or other sensitive information must be able to assist in the detection of password compromise.

The remainder of the paper will briefly describe the major features of the guideline.

4.0 System Security Officer Responsibilities

System Security Officer (SSO) responsibilities are concerned with functions that relate to both the System and to Security. Many of the responsibilities of the SSO require that the individual have a working knowledge of the operating system and be designated as a privileged user. In this combined capacity as system administrator/security officer, this individual must:

* Prior to allowing any other user access to the system, change all the standard ID/passwords when a new operating system is installed.

* Generate and distribute initial passwords to users. This requires entering a unique user ID and (if necessary) clearance level to indicate the highest classification of data that may be accessed by this user ID/password combination. Distribution of a password by the SSO to the user should be done in a convenient manner yet one that accords the password the same degree of protection given to the data it provides access to.

* On occasion, change a user's password. The distribution procedure followed here is the same as that for an initial password assignment. (Examples of reasons for making the change could be that a user has forgotten a password or a password compromise is suspected.)

* Maintain a current user ID/password database. Users having no current need to use the system should be removed as soon as possible.

5.0 User Responsibilities

The key to effective data protection via a password system is the user. The user must be properly educated on its use and on the need for "password privacy". Some steps users are advised to take include:

* Telling the password to no one

* Remembering the password and not writing it down

* Changing the password when required or when suspected password compromise has occurred

* Protecting the password from view if it is displayed on a screen upon entry

* Informing the SSO when no longer using the system

5.1 Password Lifetime

The guideline advises that passwords be changed on a periodic basis in order to avoid increasing the probability of its compromise to an unacceptably high level. Though users should be able, at any time, to change their own password without SSO intervention or assistance, the proposal is made that no password be allowed a lifetime of longer than one year. Further, it is recommended that a system-triggered mechanism be incorporated that forces the user to change the password at the end of this maximum period. If the password is not changed at the end of this period it is considered "expired" and no longer recognizable by the system at login time.

6.0 Internal Storage

A system must maintain a password database and provide the capability to create, read, add, change or delete entries. Only those portions of the system that perform these functions should have access to this database. Access controls to prevent unauthorized use of the database should be implemented. Additional protection can be afforded to the password database by maintaining entries in encrypted form.

7.0 Login Attempt Rate and Auditing

By controlling the rate at which login attempts can be made, (where each attempt constitutes a guess of a password), the number of guesses a penetrator can make during a password's lifetime is limited to a known upper bound. The guideline recommends that the system include a control that limits the rate at which login attempts can be made from any one port. It is also recommended that a control be implemented that would set a limit on the number of consecutive unsuccessful login attempts that can be made by a user. When the limit is reached (the guideline recommends a limit of 5 consecutive unsuccessful login attempts) a real time notification should be sent to the SSO or the ADP system operator. In addition, an audit trail mechanism should also record this security-relevant event and provide it to the user upon successful login.

8.0 Password Generation

As mentioned above, this guideline recommends machine-generated passwords. Experience has shown that user-created passwords are quite guessable and therefore do not always provide the needed security. Passwords created from randomly generated character sets may be so secure that even the owner cannot remember how to access the system. The goal is to use an algorithm that creates "user-friendly" (or pronounceable) passwords i.e., passwords that can be remembered, but that will still provide a sufficient degree of security.

An algorithm for generating "pronounceable" passwords could be one that uses a dictionary of a significant number of 4, 5 or 6-character words and randomly concatenates several of these words to form a "pass-phrase" (i.e. a "user-friendly" password). The algorithm should accept as input "seeds" that will ensure that the algorithm will not regenerate the same series of passwords each time the generation algorithm is used.

9.0 Password Encryption

Access control mechanisms, such as discretionary and mandatory access controls, afford some protection to the password database by allowing only specific processes (e.g., login, etc.) to access it. In order to provide additional protection, the guideline provides the suggestion that passwords be maintained in the database in an encrypted form. The encryption algorithm used for

this purpose should be sufficiently complex so that analysis of it would fail to discover any computationally feasible means of inverting the encryption function.

10.0 The Relationship Between Lifetime, Guess Rate, and Password Space

The probability that a user's password will be guessed sometime during its lifetime is a major concern for systems requiring some degree of password security. The lower the value of this probability, the higher the security of the password. Other parameters that affect password security are the password lifetime, the rate at which guesses can be made and the password space.

The guideline provides a discussion of the mathematical inter-relationship of these factors and provides suggestions on how to arrive at, say, the appropriate password length given the characteristics of the password space and alphabet size.

11.0 Conclusion

In conclusion A Guideline On Password Management provides a set of good practices for the design, use and maintenance of an effective password system. Suggested ingredients include:

* An algorithm to generate user-friendly passwords.

* An encryption algorithm and/or access control mechanisms to protect the passwords while in storage, during transmission, and while in use.

* Options to permit the System Security Officer to enter new user passwords and to change any user password.

* Users' capability to allow them to change their own passwords, but no one else's.

* System checks to ensure that a changed password is not the same as the password being changed.

* System checks on the entered user ID/password combination to ensure that it is the identification/authentication of a valid user.

* System accounting of login attempts and control over the time interval and tally of incorrect logins.

* System maintained audit capability.

* Where required, system retention of user clearance level for each password entered by the SSO.

* A maximum lifetime for passwords - established and used by the system to ensure that a password can not be used beyond a predetermined period of time.

COMPUTER SECURITY FOR TODAY

Bernard Peters

It is my pleasure to talk to the makers and shakers of computer security. At times, you seem a little slow delivering the security products that I need. But at other times, you are brilliant and very supportive. I will tell you about a lot of detailed wishes, which I believe will increase security without increasing your costs.

Let me propose a trip into the future for you. Let us suppose that all the efforts discussed at this conference are a complete success. We have a true TCB, at the A1 level. It is available for purchase, it has been purchased and it is installed at your site.

What do you need in addition to the TCB? Of course you need a system security officer. The security officer should be well-trained, knowledgeable about the work of your enterprise, and equipped to keep your system secure. This will include a terminal with a trusted path, an audit trail with tools to analyze it, and the administrative tools to keep the list of authorized access authorities and the lists of users accurate and complete.

You will also have a secure space for the system. It will be appropriately protected against intrusion, theft, fire, and other detracting activities. The communication lines will be suitably protected or judged to be sufficiently secure.

The security officer will have available some on-line tests to verify that the hardware and software mechanisms which provide security are working. These tests may be run at will or off the clock.

The executive level of your enterprise will have determined what information is to be protected and will have promulgated the policy needed to protect the data. Everyone working in the enterprise will be expected to support the protection efforts. It will also include the training and education of the users in the doctrine for correct and secure use of the system.

All users will have a personal identification, and a pass phrase known only to themselves. There will be appropriate means to share data without losing the identity of each user. Passwords will be changed at some period not exceeding 90 days, with the user free to change at will. Failure to change a password when outdated will be brought to the attention of the security officer.

The name of the security officer will be known to all users. Users will be encouraged to report all security incidents such as unusual use or mis-delivered data. Yes, it will still be possible for users to missend data. There will be someone to call to report security problems or concerns during all hours of use for the system.

Tapes, disc packs, floppy disks and any other media will be properly labeled with external labels as well as machine readable internal labels. The software will respect the labels to assist in avoiding error.

External connections to the system shall be made carefully. Where reasonable, the links shall be encrypted. In other instances, strong management of the connection will assure that the user is most probably legitimate. Of

course, the full power of the pass phrase and login machanism shall be employed to additionally screen the external user. The external user may be restricted in the available activities. In any case, the external user will receive special attention in the audit trail.

Maintenance will be done by trusted persons. The security officer will have available independent tests to assure the configuration is unchanged with respect to security.

With all this in place, you have a secure system. You can feel comfortable. Except for the inevitable human errors your system will operate securely.

WE CAN HAVE THIS TODAY!

If all the above were available today, if would significantly enhance security. These mechanisms are possible without the existance of an AI system. And it is criminal that they are not in place. The current users of computers have paid for the industry. They deserve the best possible security for the information and processes they have entrusted to the computers.

Let us now examine what can be done today, or soon, to obtain a secure computer system.

The terminal is the interface to the user. The terminal should present a secure and security conscious interface to the user. For instance, do not invite vandalism. The terminal should not say login, but should announce, "This terminal secured at 00:00:00 by mruser. Push panic twice for service." This will lead the user to execute a non-interceptable procedure to call for the login procedure. This defeats certain potential threats such as the false login program. This is useful even if the terminal does not possess a non-interceptable key. It helps those in the area to manage access to the terminal. It complicates the efforts of anyone attempting to build a false login program. The login program should have a timeout feature. If there is no typing for as little as one minute, then the terminal should be returned to the secured state.

Of course the terminal should be timed out for non-use. At five minutes of non-use, defined as not a single character typed, with no output either, then a one minute warning with bell should be issued. If there is no response, then the terminal should be secured, with the event logged into the security log. Timeout should be as non-destructive as can be. It should permit background tasks to complete, and if feasible, it should push the foreground task into a sleep state for possible recovery and continued use. But, it should definitely shut the terminal down.

What should the user of a terminal see and do before receiving service? As mentioned, the user should ordinarily find the terminal in the secured state. The user should review the information on the screen. The identity of the last user and the time of use should be reasonable. Service is called for by hitting panic of another non-interceptable key. All other keys are ignored until the selected panic key is pressed. The system then asks login. The user should furnish an identification of some length. I recommend seven alphabetics made up of the initials and first five letters of the last name. I recommend that each user use the same personal identification on all computers at the enterprise. This will require a mechanism to resolve duplication for the larger enterprises.

Under no conditions should anyone depend upon the identification of anyone being kept secret. The pass phrase or password must be the secret element.

The login program should give no hint to the entering user as to whether the supplied identity is acceptable. The login program should however permit the entering user to correct mistypes or even cancel the line already typed. Under some UNIX implementations, if the first letter is capitalized, then the whole session is considered to be capitals. This makes passwords with lower case in them not work. But there is no way to cancel the login except submit enough bad logins to cause termination for excessive bad logins or else ignore the terminal until it times out and secures itself.

After the identification is received, the terminal should request the password. Pass phrase might be a better description. The user should type to a non-echoing screen. If echo cannot be suppressed, then perhaps the echo can be a non-useful character such as a dot or question mark. If the character typed must be echoed, then added effort is required, I will get to that case shortly. As the entering user types in the password, the system should encrypt each character into a binary password. The characters that make up the password should not be collected as a plain password. It is not necessary and it constitutes a minor insecurity. It is sufficient to encrypt them into the binary password as submitted. Only upon getting the character designated as the end of line, new line, or carriage return, should the binary password be used.

The number of characters typed should be counted. If the number exceeds forty, then the terminal should be secured, with a security log entry being made. This is to deal with the inevitable sending of data to a login sequence.

Every user, without exception, should enter the system through one approved login sequence. This includes superusers and operators. After appropriate identification, any privilege may be granted, but until identified, any entering user must be viewed as a hacker or vandal to be kept out of the system. Identification is not complete until the authentication is accomplished.

On some systems, if a user fails to authenticate after three tries, the system locks up, disables the terminal until it is restored by the system security officer or an operator. I dislike such denial of service. I recommend the following strategy. When opening the login activity set a value to zero. Upon failure to authenticate set the value to one if zero, otherwise double it. This represents the number of seconds to delay before asking the login series again. Announce on the screen that there is a security delay, and how long it is. As you may know there are foolish people in the world who disable terminals to annoy others. Using this scheme, a vandal must attempt to login for 59 minutes to disable the terminal for an hour. The need to sit at the terminal will be a strong deterrent.

Occasionally one hears of systems where the systems programmers or superusers have short login sequences. In my opinion, such systems are subject to well-deserved disaster.

If the terminal cannot blank or block the password as it is being typed, then the system should ask for the password to be typed into a screenful of random characters. If possible the screen should be filled with characters of mixed video to include blinking characters. If the terminal provides for automatic stepping from field to field, then such features should be used to scatter the password throughout the screen. The screen should be erased as soon

as possible after transmission. Paper oriented terminals, which cannot suppress echo, should not be used.

What should the user be told by the system as he signs on? After the user has identified and authenticated, the system should identify itself and tell the user which terminal the system believes the user is entering from. Sometimes users forget which terminal they are using. The system should then tell the user if he is logged on at any other terminal. The user has the highest interest in preventing improper use of his identity and will call for inquiry into any unusual use. This assumes that group identity and group passwords, or sharing of passwords is prohibited, as it should be. On ocassion, a user will use more than one terminal located together in a group. This is acceptable.

Next the user should be told of the date amd time of this account's last login. This gives the user an opportunity to consider whether the recent usage is reasonable. If the user has been on vacation or travel, and there has been use of the account then there is a basis for concern. If recent usage is not shown, then there may have been unacceptable system restoration and the user must be concerned about the timeliness of the data. There is a lot of punch in this idea. The user is concerned about his account. He will watch the date and time of last use to safeguard his billing and accountability. The system should provide as much information as can be found. This could also include the last terminal of use, pages printed, connection to other computers, or other interesting usage.

The user should be told his security level and privilege level. This will remind him of his responsibility. It may also lead him to request more appropriate security levels. We don't need users who are stopped because their security level was wrong. If the user knows his security level, he can respect the restrictions thereon.

Persons authorized superuser or special privilege should obtain such privilege with a specific overt act, after signing on. Such authority should not be given or obtainable by the first act of signing on. On some systems, in addition to having the users identity on a list, the user must know the current superuser password to acquire privilege. In any case, any attempt to obtain privilege, whether successful or not, should be logged to the security log.

After the user has been told of duplicate sign-on, last sign-on, and current security and privilege level, then an appropriate notice to change password, if due, should be given. Some systems force immediate password change if due. Other systems permit the user to continue, but remind on every sign-on. I find the forced change now to be unfriendly and not necessarily more secure. The strategy to be used is site dependent, but a friendly call by the security officer will also get the password changed. The system can report who is overdue by an unacceptable time or number of sign-ons.

The system should be generated to put all users in the mode of least privilege. Whenever a user generates a file, such file should be restricted to the user only, unless the user takes specific action to make it available to others. Some systems are configured to make all files public unless the user takes specific action to restrict access. I denounce such systems and those who configure them.

The user may make some errors that might be security significant, such as seeking privilege not authorized, or access to data not authorized. The user

should receive an appropriate error reply, and the attempt should be logged in the security log. The attempt should not be simply forgotten. Nor should the terminal and user be locked out. Denial of service is appropriate in only the most grave circumstances. There will be many errors prior to any genuine espionage.

The system should be able to restrict the user to a non-programming mode. This would permit the user to make use of existing programs to make inquiry and even updates to a data base, but would not permit programming attacks on the system. Such a system resriction of use should be carefully checked to make sure that it is complete. Under UNIX, it is sometimes possible to escape from the restricted shell. Such escape must be prevented, if unacceptable. The decision to allow users to access data bases should be made by the user and owner of the data base. It should not require the intervention of the security officer. The security officer must not be looked on as a conscientious clerk.

The security officer has one principal duty, to review the audit trail and security log. The security officer must detect and pursue misuse of the system. The security officer may give advice and admonishment to users, but is not responsible for training them. The security officer represents management and should have access to an effective level of management.

I have mentioned UNIX several times. It is a system of some importance, and you should be concerned with the security of your implementation. It is possible to have a secure UNIX, but it takes some effort. I recommend that you make the effort.

There are some security features which UNIX does not have in most implementations. Some features are called for by the TCB Criteria, and some are a good idea anyway. First, let the clock be a hardware feature. I find it amazing that I can buy a personal computer that knows the date and time even after power down, but that a large computer system can have its data base damaged because the operator typed in the wrong date! Please provide a clock and calendar card. It makes the security log more valid and valuable.

I prefer that the above mentioned label and CRC be part of the data, prepended to the actual data. This would permit variable length data items. Of course it is possible to create conventions to append the label and CRC, but I prefer prepending. Most sorts can readily ignore the first n characters of each data item. It is hard to deal with data items that are short and might have the label and CRC in the sort field. Creating the label and CRC apart from the data item seems dangerous. Don't do it.

You should furnish the label conventions and CRC generating routine. You should provide supporting routines to check the CRC and label. Your documentation should help the user decide on four or eight byte CRC's, explaining the confidence levels to be attained. Recommended methods for dealing with CRC failure should be presented. I hope some standards group will get on this so that we can apply a CRC at first creation of a data item, to be used until the data item is destroyed, on every system that handles it. That is a big wish.

It has always been possible for a computer vendor to offer a secure computer system. I find it amazing that not all vendors do. Security is as important as accuracy, as reliability, as ease of use, and as economy. The computer vendors have been requiring the users of their systems to hire saints

to do their work. How dare they? They certainly have not hired only saints to do their software. Improved security is in everybody's interest.

But, you say, there has been no method to prove that a system is secure, until the Evaluated Products List process was created. If a system is on the Evaluated Products List, every user will consider the system because it may meet the user's security needs. But the system must meet performance and fiscal criteria. Then the user will want to test it. Yes, test it! And the user will expect the vendor to provide the security tests, just as the vendor now provides performance demonstrations.

Vendors can provide security tests that are independent of the software that provides the operating system and security relevant utilities. Such tests, if well documented, will provide the customers with the ability to choose secure systems. They will permit the user or security officer to test the system as needed. Such tests should be available on an on-line mode. If the system is to be used seven days a week, 24 hours a day, it should not be necessary to bring it down for security tests.

What are such tests to demonstrate? That every security assertion is supported by software and hardware? It is easy enough to submit a batch of bad login attempts, bad file references, attempts to reach out of bounds, etc. It is hard to do this, and to keep the reaction correct. Log such failures and security relevant attempts, but please log as submitted by "SECTEST" so that the security log analyzer can easily dismiss them.

Give the security officer some tests, to run as he sees fit, either off the clock, or as needed. There are some security features that cannot be reasonably tested on-line such as secure start-up and secure power-fail. All reasonable tests should be performable on-line.

A fair number of security tests should be available to each user. The user deserves to have confidence in the system also. We have implemented a program on UNIX called "Expose." Expose tells the user which files of his are readable or writeable by the public, or by his group. Many times when we have given this to a user, the user was unhappy with the report. Quite often, the user felt the need to change some of the permissions. Such tools help the user protect his data and thus the system is more secure.

Why not provide the user with good residue management? It is not a big deal to assign released file and memory space to a list to-be-freed. The movement of such data from the to-be-freed list to the free list can occur when the operating system gets around to erasing such areas. Worried about performance on the real-time level? Give real-time activities the right to have error free real-time activities.

The audit trail, as well as the security log, must be reviewed by a competent security officer on a daily basis. There must be tools to condense the mountains of detail to items of interest. It is unreasonable to expect the security officer to read masses of detail and recognize security significant events without aid. The vendor should furnish such tools. The scan for security items is no harder than the scan for performance items.

Using the computer to scan the audit trail has advantages. The computer will be consistent. It will work any relationships regardless how tiresome the work, or how seldom it has produced a result. The computer can study cross

relationships that bore humans to tears or sleep. The computer has no friends, it will turn anyone in. Some sites have installed some automated audit trail review. It is quite successful and useful. It is important that the results go to a security officer and are not used to generate automatic messages of rebuke. Only humans may admonish humans.

Vendors, please put some security on your software. Give the user an independent means to check the source and object code for validity. This could be an overall CRC. There is in the literature a number of security attacks that are based on false issuance of fixes. You can easily prevent this. Do it. It is embarrassing if not dangerous, in a sales sense, to be named as the system which suffered a major loss. If you or your best people don't know how to do this, call the Computer Security Center or myself.

Under UNIX, as well as some other systems, there is a tendency to permit the superuser umlimited power. I suppose that it can be argued that such unlimited power is required in certain cases. It is not, however, required in most of the cases where such authority is given. For good computer security, the users must have the superuser disassembled. There can be serious damage inflicted on a system and its users by an errant use of superuser powers. Let us have some breakdown into more reasonable partitions. An operator partition, a librarian partition, a source code manager partition, a system administrator partition, and a security officer partition are a minimal prudent set.

The lack of item and file locking in UNIX is a bit of a scandal, but I am sure it will soon be fixed.

SUMMARY

I assert that many of the system usage features and operating procedures needed to successfully use an A1 TCB are already known. Further, these features if implemented today would give a large measure of needed security. Some sites have been depending on some or many of these features for years. Much is known about these measures. We need to implement them on most of our systems as soon as is practical. There is a lot of hard grubby work to be done, even if a TCB were available right now. Let us get on with it, such work can give us a lot of security very soon.

MICROCOMPUTER-BASED TRUSTED SYSTEMS FOR COMMUNICATION AND WORKSTATION APPLICATIONS

Dr. Roger R. Schell
Dr. Tien F. Tao

Gemini Computers, Incorporated
P. O. Box 222417
Carmel, California 93922

INTRODUCTION

Gemini Computers, Inc., is developing a family of high-performance secure computer products that provide reliable security for both military and commercial users while providing the utility and effectiveness needed for advanced applications. The hardware, based on the Intel iAPX286 processor, has been designed and pre-production units constructed and tested. The Gemini Multiprocessing Secure Operating System (GEMSOS) is designed to the Class B3 security requirements and a developmental evaluation by the DoD Computer Security Center has been initiated. The implementation, almost entirely in Pascal, uses strictly-layered modules that are incrementally integrated to form a running subset. As of the writing of this paper, the capabilities for multiprogramming, multiprocessing, core management, secondary storage management and device management are integrated and running several demonstrations.

BACKGROUND

We believe that we will in the future see a dramatic increase in the need on a wide spread basis for trusted computer systems. The required technology exists and action now by users, government and industry will insure that adequate products are available and are used.

Until recently there was not a pressing widespread need for a high degree of trust in the hardware and software internal to the computer itself because adequate security could be provided by severely limiting which individuals had physical or electrical access to the machine. However, the proliferation of computers and the networks to interconnect them means that in the future the potential users of many computers will include individuals not authorized access to much of the growing amount of valuable and sensitive information they contain. The exploits of the hackers serve to illustrate that they are in fact potential "users" of many computers whose owners failed to recognize them as such. At the same time the experience of both the hackers popularized by the press and the sanctioned "tiger teams" illustrate what most computer professionals already know -- that the hardware and software of many of the popular systems are highly vulnerable.

Fortunately more than a decade ago some very powerful research results were produced by (mostly government-sponsored) computer security work in universities and research centers. In particular the

GEMSOS that locates readable data and code segments in the local memory of each microcomputer.

The Gemini system supports a variety of storage and I/O devices through interface boards directly connected onto the Multibus. Under GEMSOS control, microcomputers share all devices interfaced on the Multibus. The system supports any combination of up to four disk drives which include 85 Mbyte fixed Winchester disks, 11 Mbyte removable Winchester disks, and 1 Mbyte double-density double-sided floppy diskettes. Non-volatile memory is available for "core resident" applications. Each RS-232 serial I/O interface board handles 8 ports.

The system includes the Gemini bus controller, a real-time clock with battery, data encryption device using the standard NBS-DES algorithm, and a unique system identifier. The system also contains a non-volatile memory for storing system passwords and encryption keys.

Security, Integrity and Privacy

Both hardware and GEMSOS security kernel software support security, integrity and privacy to meet the Class B3 requirements of the DoD Criteria. The iAPX286 microprocessor provides hardware support of memory management, combining the CPU and MMU on the same chip. The iAPX286 supports four hierarchical privilege levels (commonly called protection rings) for protection and mediation of all memory and I/O references.

GEMSOS takes full advantage of these hardware security and protection features. All information stored in a Gemini system is contained in discrete logical objects called "segments." Every segment posesses attributes such as security access class and access mode. GEMSOS supports both sensitivity and integrity access classes, each with 8 levels and 24 compartments, to enforce non-discretionary (mandatory) security policies. GEMSOS also provides an environment for enforcing application-specific discretionary security (need to know).

Additional security support is provided by the hardware encryption device for the standard NBS-DES algorithm. Each hardware unit has a unique master key and system identifer. This is used to insure the trusted distribution of GEMSOS releases, to control software piracy from removable media by limiting execution to specific hardware units, and to encrypt the information stored on removable media such as floppy diskettes. Encryption can also be used by applications to prevent unauthorized access to transmitted data and authenticate the integrity of received data.

Concurrent Computing

The multiple microcomputers in a Gemini system are capable of multiprocessing as well as multiprogramming. Depending on the requirements of a particular application, the GEMSOS can multiplex processes onto a single processor or distribute them on several processors to support combinations of parallel and pipelined pro-

cessing. The emphasis is on performance and throughput for the application, not just the theoretical "efficiency" in the use of the individual hardware module.

Gemini's approach to concurrent computing does not depend on the availability of concurrent programming languages. Concurrent computing is based on the combined support of well-developed sequential programming languages, the concurrent GEMSOS, and the Gemini homogeneous multiple microcomputer hardware architecture. GEMSOS synchronization primitives manipulate objects called eventcounts and sequencers to support communication and synchronization among processes. Sequential progamming languages can use these GEMSOS primitives in concurrent computing application programs. GEMSOS also provides priority scheduling, access to a real-time clock, and application controlled processor allocation, allowing programmers great flexibility when designing and developing concurrent and/or real-time applications.

Self-Hosting Software Development Environment

The Gemini multiple microcomputer system is a self-hosting environment for software development using the popular CP/M-86. Any programming language that runs on CP/M-86 potentially can be used to develop concurrent computing and multilevel secure application software. Gemini is currently supporting the following languages: Pascal MT+, JANUS/Ada, PL/1, C and Fortran. Additional software development tools are provided by several utility programs supplied as a part of the Gemini system.

Modular Expansion amd Configuration Indepemdence

The Gemini system is built on the IEEE 796 standard Multibus and can therefore use literally hundreds of different products developed for Multibus systems. For those products already supported by GEMSOS, expansion and growth of a Gemini system can be as easy as inserting a new board into an empty Multibus slot. Gemini configurations include up to 26 slots. The GEMSOS software has been developed to support modular expansion and configuration independence. The system automatically adapts to the available microcomputers and memory on power-up.

In addition to the suppported products, Gemini Computers, Inc., is prepared to respond to requirements for special applications. Graphics controllers, image processors, voice processing boards, array processors, communication interfaces, data acquisition boards are just some of the special purpose Multibus boards potentially supportable in a Gemini system. The GEMSOS is highly modular and coded in Pascal MT+ to simplify adaptation. GEMSOS has already provided the "hooks" for several extensions. The serial I/O can be adapted to support RS232, RS422, RS423, and current loop physical interface standards. Parallel I/O interface boards can be used which supports the standard IEEE 488 GPIB protocol. For packet switching, multiple Gemini nodes can be connected in a 10 Mbps Ethernet local area network with GEMSOS adapted to support the DoD standard IP/TCP protocols. For circuit switching, the serial I/O ports can be adapted to support a wide range of data

communication protocols: Async, Bisync, Bitsync, HDLC, SDLC, ADCCP, etc. Gemini will continue to evaluate additional languages and develop special software interfaces to support languages compatible with Gemini systems.

GEMINI SECURITY ARCHITECTURE

The Gemini security architecture draws from and builds on the positive experiences and results of previous research and developments. For example, the architecture is deliberately designed to be "entensible with respect to security" as described by Schaefer [3]. In particular we demonstrate his conclusion that,

> "Given strict hierarchical layering ... along with a strict integrity policy mechanism such as the ring mechanism, it should be possible to extend a system through the addition of new adjacent domains. The mandatory policy and its implementation is essential; most of the other capabilities, including discretionary security and selective audit, can definitely be added on to such a base."

Security Domains

The iAPX286 hardware supports four protection levels [4] that GEMSOS uses as classical protection rings to create the dominance domains to enforce the security critical layering of the system; as in Multics, these will be referred to as Ring 0 through Ring 3, with Ring 0 the most priviledged ring.

Rings 2 and 3 are outside the security perimeter, and their use is left largely to the customer. We would anticipate that Ring 2 would be used for common services employed by many users, e.g., a data base managemant system. Ring 3 would typically be used as purely an applications layer for programs and data provided by individual users.

Rings 0 and 1 contain the implementation of the formal security policy model -- in particular the Bell-LaPadula Model [5] is used. However the nature of the split between the two rings provides extensibility. Ring 0 contains a distributed kernel that implements the essential foundation for non-discretionary (viz., mandatory) policy for which the model provides a strong set of provable properties. The kernel must provide extended virtual machine that specifically support both asynchronous processes and segmented address spaces. The kernel virtualizes processors, all levels of storage, and I/O, as well as creating virtualized objects -- processes, segments, and devices. Ring 1 contains a supervisor that implements the discretionary policy and other security capabilities that can be added to the mandatory controls. The supervisor is designed to be built on the kernel, using the virtualized objects to provide the usual functions of an operating system, such as a file system.

security kernel technology [1] for engineering secure systems, with the associated formal mathematical security policy models as a definitive basis for their evaluation, provided a clear foundation for achieving systems where the hardware and software controls can be trusted to protect computed information from unauthorized modification or dissemination. Today this still remains the only available technology for demonstrably secure computer systems of practical proportions. During the past decade the maturation of this technology and the improvements in software engineering and microelectronics have made the security kernel technology practical and affordable, although not yet widely assimilated.

The future of trusted systems depends on vendors offering suitable trusted products, on (probably) government providing objective standards of trust and on customers recognizing the need and employing the available products. Several security kernel-based commercial products are or will soon be available for systems ranging from large-scale, general-purpose mainframes to microcomputers such as the Gemini system. The U.S. Department of Defense (DoD) has recently published an objective set of Trusted Computer Systems Evaluation Criteria [2] with seven distinct levels of trustworthiness and is expected to publish the results of its evaluation against these criteria of selected products. It is not yet clear how widely these criteria will be embraced, especially in view of the risk that some popular products may not fare well. Furthermore, with virtually no laws, regulations, policy or practice to mandate or even encourage meaningful hardware and software controls, some believe there is little incentive for management to buy the trusted products that are available until there is a disastrous "three-mile island" of computer security.

However, we are persuaded that as products become available that are both practical and secure, users will step up to their responsibility to protect the information entrusted to their machines. This paper will describe the capabilities and structure of a new family of products that responds to this need. Our contribution is not one of fundamentally new ideas, but rather of bringing together into an integrated whole the available hardware components, software engineering techniques, and trusted systems technology, making the result available as an off-the-shelf product.

THE GEMINI FAMILY HIGHLIGHTS

The Gemini family of microcomputer systems provides a powerful combination of multilevel security and multiprocessing capabilities. The adaptability of the Gemini Multiprocessing Secure Operating System (GEMSOS) makes the Gemini systems attractive as a trusted base for a wide range of concurrent and real-time computing applications such as command, control, communication, intelligence, weapons, networks, and office automation.

Tightly coupled multiple microcomputers communicate through shared memory segments to provide high throughput performance. Up to 8 powerful Intel iAPX286-based microcomputers are modularly connected on the same Multibus to increase processing power. Processor and memory modules are interchangable. Bus contention is minimized by the

The Supervisor: Ring 1

The functionality of the supervisor will in practice vary substantially for Gemini configurations. For example, when used as a trusted network interface unit that serves as a controller for end-to-end encryption as proposed by Rushby [6] the supervisor has minimal security responibility.

On the other hand, in a configuration as a workstation discretionary security is provided in the supervisor ring. A supervisor can implement a file structure out of segments with a file made up of many segments. The discretionary security attributes and other attributes of a file can be stored in a separate segment. The non-discretionary security of the files is guaranteed by the underlying GEMSOS kernel. While our first file system is an emulation to support the CPM-86 software development environment, we expect that several different supervisors will be developed both to provide an interface to other existing application software and to support the specialized discretionary policies of some users.

User authentication, system security officer and audit functions are also provided in the supervisor ring. Distinct trusted subjects (processes) are used to support those supervisor functions related to the non-discretionary policy. The Gemini hardware includes non-volatile memory specifically to store passwords and other authentication data. Each user must log on to the system with his name and password for authentication. In a floppy disk environment, eack disk used in the system has an access class associated with it. The relationship between the user's access class and the disk's access class is checked at log in time. A user will not be allowed in the system unless his access class is compatible with the disk's access class. This security cannot be bypassed by changing to different disk. Any attempts to use an unacceptable access class will be detected and recorded for audit purposes.

Similarly, when appropriate to a given application, the use of the DES encryption hardware will typically be controlled by a trusted process in the supervison ring. This process has access to the master key and system identifier for features such as encryption of removable media and limiting the execution of imported software to only legitimate hardware units.

The Distributed Kernel: Ring 0

The interface to the GEMSOS non-dicretionary kernel implementation is a set of "virtual instructions" in the classical extended machine sense -- these consist of (1) hardware instructions (2) calls to the distributed kernel and (3) interprocess communication to the non-distributed kernel, viz., Ring 1 trusted subjects, if any. Now what is meant by a distributed operating system or distributed kernel is one that is distributed in the address space of the user processes. In practice, this means that the distributed kernel is a distinct domain in the process; it is invoked by the mechanism to "enter" a domain -- for the iAPX286 either a "cross-ring" procedure call or a "trap". In contrast, the non-distributed kernel is invoked by inter-

process communication to the distinct process that may execute simultaneously with the sending (viz., invoking) process.

There is a very strong property of the distributed kernel that significantly simplifies assuring its correct behavior at the interface: it forms what can be thought of as logically a critical section with respect to the invoking domain. This property derives from the fact that a process has only a single execution point, and when it is executing in the kernel domain there is absolutely no way it can be simultaneously executing in the user domain. Thus many of the difficult issues of concurrency can be avoided merely by using the strong notion of a process. This notion permits the kernel to be both interruptable and reentrant so that it does not form an intrinsic performance bottleneck.

The kernel is responsible for enforcing mandatory access limitations; that is, the kernel provides the mechanism for supporting the non-discretionary security policy. The kernel can support any policy that can be expressed by a lattice of access classes. Every object -- process, segment, or device -- has a nonforgeable label that denotes its access class. This non-discretionary security label has been assigned parameters such that exactly one module knows the interpretation of this label in terms of a specific policy. Thus, not only does the kernel support a broad range of security policies but only a single module has to be tailored to support a particular policy.

The mandatory policy constrains not only the interface but also the detailed design and implementation of internal state variables. One significant problem is preventing indirect information channels between processes with different access classes. For evaluation internal state variables, such as shared resource tables, are assigned an access class, and it is confirmed that the design ensures that values will not be reflected to processes with an inconsistent access class. The most apparent result is that the success code (returned in response to the invocation of kernel interface primitives) reflects the state of the per-process virtual resources, not the shared physical resources. The same confinement problem requires a non-exclusionary approach to provide secure synchronization between processes of different access classes. The interprocess communication provided by Reed's event counts and sequencers [7] provides the solution to this "secure reader-writer problem".

The design goal for GEMSOS was to meet the Class A1 DoD criteria. An initial version of the top level specification was prepared in a variant of the GYPSY formal specifcation language. However, the lack of access to the required formal verification tools led us to design the current version of the product specifically to meet the Class B3 requirements. However, we have continued to be alert to the Class A1 requirements and believe that the current architecture canbe extended to meet these requirements.

NON-DISCRETIONARY DISTRIBUTED KERNEL IMPLEMENTATION

The implementation of the non-discretionary, distributed kernel has applied lessons learned from previous security kernel developments, and especially the previous experience with the iAPX286 microcomputer [8]. One significant change was the use of the Pascal MT+ language. This selection was driven by the desire for a strongly-typed language to ease systematic evaluation and by the need for certain features essential for engineering a distributed kernel, e.g., reentrant code, stack oriented procedure invocation, ability to reference separate per-process and shared data segments, controlled interface to assembly language routines, and a production-quality compiler and linker. There were few acceptable choices, and the performance price of the Pascal MT+ choice is still being assessed. There was inadequate support for inter-segment calls; we have (reluctantly) designed our own intersegment linkage mechanism.

The implementation is proceeding as a rapid prototype to permit empirical measurement of performance bottlenecks. This prototype has substantial design documentation to allow detailed examination of the internal security structure as well. It is our intent that this will permit us to be responsive to suggestions during the developmental evaluation by the DoD Computer Security Center while providing an early and stable interface for users of the system followed by a smooth transition to the fully evaluated product.

A significant feature is the initialization approach to enhance configuration independence. The system's first response to the hardware bootload signal from an operator or at power-up is to determine dynamically the available processors and local and global memory. This configuration information is then used to complete what in many other systems is a static system generation procedure. This initialization places the system in its "secure initial state" required by the policy model and thus is properly regarded as part of system generation and not part of the security kernel, per se.

As noted previously, the GEMSOS Ring 0 rigorously enforces the non-discretionary security policy and is the distributed kernel that provides the fundamental, extensible foundation for all configurations of the systems. Therefore, it is of paramount importance that its implementation provide a high degree of assurance in its correctness. A principle design property that has helped to keep the security kernel simple and understandable is the loop-free structure of the modules. The loop-free design supports the software engineering concept of "information hiding" [9]. As a result, GEMSOS does not functionally have any global data structures. The kernel is internally organized into eleven distinct layers that will be described below.

In practice we have been quite doctrinaire in enforcing the loop-free structure for the layers. While many operating systems claim to be modular or well-structured, our testing and integration approach empirically validates this claim. The modules are integrated and tested from the bottom up. As each module is completed, an accompanying controlled and reproducible test set is prepared. In the GEMSOS prototype implementation the major functions and error condi-

tions visible external to the module are tested, and a careful audit and accounting is maintained of which sequences of code are tested; as the production version is integrated, the test will be reviewed and augmented to insure that each sequential code sequence is executed by some test. For the actual execution of its tests each module is integrated with those modules below it that it invokes. This integrated unit is loaded and run as a functionally intact, but obviously limited, operating system subset.

Each module is effectively in its own dominance domain, although there is no supplemental domain enforcement from the hardware. The following briefly describes each module in the order in which the layers occur, from the bottom up.

Core Manager

The core manager provides the interface to and "hides" the information about the details of several unique charteristics of the processor. This module manages the local and global description table that control the memory management functions of the processor chip. It also provides a controlled interface for interrupt processing.

Inner Traffic Controller

Processor multiplexing has two layers, similar to those proposed for Multics [10]. The bottom layer provides essentially what some have termed a "separation kernel" [6]. Each physical processor has a fixed number of "virtual processors" that are multiplexed onto it by the bottom layer that we call the inner traffic controller. Two of these virtual processors are dedicated to system services: an idle virtual processor and a memory manager virtual processor to manage the asynchronous access to secondary storage devices. The remaining virtual processors are available to the (upper level) traffic controller. The inner traffic controller provides primitives for synchronization between virtual processors. In terms of traditional jargon, this means that the inner traffic controller provides multi-programming by scheduling virtual processors to run on the processor with which they are (permanently) associated.

Note that this structure implies that the security kernel can be (and is) interruptible, viz., is not a critical section; however, the inner traffic controller itself is not totally interruptible. In addition, this layer provides all the multiprocessing interactions between individual physical processors, using a hardware "preempt" interrupt. An additional benefit of the strict layering in the design is that the existence of multiple processors is visible only at this lowest level. Thus, the multiprocessing adds no difficulty to the design of the rest of the kernel.

Inner Device Manager

This module provide the hardware specific drivers for the user accessible devices. The devices include the serial I/O ports, the DES encryption hardware, the unique system identifier, the real-time calendar clock and the non-volatile memory for the system password and encryption key storage.

Non-Discretionary Security Manager

This is the only module that interprets the stored access classes in terms of a specific security policy. Subjects and objects in the system have both a sensitivity and integrity access class. The default version supports 8 hierarchical levels and up to 24 compartments. However this module is designed so it can be easily adapted to alternate policies, for example, to support community of interest separation for hundreds of distinct categories.

Secondary Storage Manager

This module provides the interface to and manages the use of the fixed Winchester, removable Winchester and floppy disks used for permanent storage of segments. It makes the non-discretionary security checks for removable volumes. This module interfaces to the Gemini disk format that provides an access class label for each segment on disk and a range of allowable access classes for each volume.

Memory Manager

This layer manages the multiplexing of the physical primary storage resources. This layer also manages the segment descriptors in the iAPX286 descriptor tables for each process. Several of the functions of this layer are executed by the per-CPU memory manager virtual processors, with synchronization provided by inner traffic controller primitives. Each processor has local memory that is addressable by only that processor. There is also additional global memory that is addressable by all processes running on any processor. The choice of the memory used has significant performance implications.

Bus contention is a major performance concern in the multiprocessor configurations, since all processors share a single bus. However, in reality only shared, writable segments need be in a global 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 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; however, 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

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.

Upper Traffic Controller

The variable number of processes (the "subjects" of the system) are multiplexed onto virtual processors defined by the inner traffic controller. Each process has an affinity to the physical processor whose local memory contains a portion of its address space at the time of the process scheduling decision. As indicated earlier, the traffic controller layer uses Reed's advance and await mechanism to provide secure interprocess communication.

Segment and Event Managers

All entries into the kernel from outside Ring 0 that refer to segments pass through the segment/event manager layer. The explicit non-discretionary security checks are made by invoking the non-discretionary security module to compare the access class labels of subjects and objects. This layer uses a per-process known segment table to convert process local names (viz., segment number) for objects into unique system-wide names. Each segment has associated with it an eventcount and sequencer; the segment numbers also serve as the names used with the interprocess communication primitives.

Segmentation. In GEMSOS, all information, both code and data, is stored in segments. A segment is a block of storage ranging in size from 0 to 64k bytes. All segments have an associated access class label that defines the sensitivity of the information and is the basis for determining the classes of processes that can access the segment and the kind of access they can obtain. Segments are typically stored on disk along with descriptive information, e.g., access class. The GEMSOS kernel interface provides six functions for segment management.

create_segment and **delete_segment**

These allow application processes to dynamically create and delete segments, specifying their size and access class.

make_known_segment and **swapin_segment**

These calls are used to gain access to segments with a desired mode, e.g., read, write, or execute. Makeknown allows the segments to be introduced in the process address space. This allows access to any segment in secondary storage, provided of course, that the process meets security restrictions. Swapin moves a segment into the primary storage allocation of a process.

terminate_segment and **swapout_segment**

These two complementary calls are used to remove a segment from a process's address space and primary storage allocation.

Synchronization with Eventcounts and Sequencers. Inter-process synchronization is accomplished through the use of eventcounts and sequencers [7], that allow asynchronous control of cooperating processes. Communication between processes is provided by the use of shared segments. Any type of synchronization can be implemented through the use of eventcounts and sequencers. Inter-process synchronization and mutual exclusion are easily done with these operations, as well as other more complicated synchronization schemes. The GEMSOS kernel interface provides four primitives for process synchronization:

advance, await, ticket and **read_evc**

An eventcount has a name and a value. The GEMSOS associates an eventcount with each segment and identifies it by a segment number. The eventcount value is initially zero at segment creation and can only be incremented by a positive value of one. Therefore, it has only positive non-decreasing values. Eventcounts are used for process synchronization and to control the relative order of execution of processes. They are incremented by the kernel call "advance", and a specified value is waited on by the kernel call "await". The event-count value can be read by the kernel call "read_evc".

A sequencer also has a segment number as a name and has a value. It is used to assign an order to events occurring in the system. A sequencer, like an eventcount, is a non-decreasing integer variable that is initially zero. "Ticket" is the only kernel operation allowed on a sequencer. A ticket call returns a unique value which is the current value of the sequencer for the specified segment and then increments it by one. A sequencer can be used with an eventcount to effect mutual exclusion.

Upper Device Manager

This is the module for the interface to I/O devices from outside Ring 0. This module checks the non-discretionary security attributes for the devices. The GEMSOS kernel provides four functions for device management.

attach_device and **detach_device**

They are used by application processes to gain and terminate access to devices.

read_string and **write_string**

They are used to transfer data to and from the serial I/O ports as well as Gemini devices such as the encryption hardware.

Process Manager

This module provides the interface from outside Ring 0 for the management of the application process structure. The GEMSOS kernel provides two functions to dynamically create and delete processes:

create_process and **delete_process**

Gate Keeper

A process in the application or the supervisor ring invokes a security kernel function using the hardware supported functions for changing the execution domain to Ring 0. In effect, this is a "system call" instruction that causes a trap, and the gate keeper is merely a trap handler. Most parameters and return values are "passed by value"; this simplifies security validation. The protection parameter verification instruction of the iAPX286 provides further assistance in the passing of parameters to the kernel. The gate keeper merely calls the particular module that corresponds to the requested function.

SUMMARY

The Gemini trusted multiple microcomputer base supports a wide range of security and concurrent processing applications. We believe it is a particularly attractive and cost effective base on which customers can develop a variety of advanced capablilities.

In modern communications system a dedicated, "core-resident" configuration is well-suited as a secure front-end or interface processor. The flexible multiprocessor support provides a low risk for at the same time meeting demanding throughput requirements such as those encountered in high speed packet switching or end-to-end encryption control. The integral DES encryption hardware makes it also attractive for "guard" interfaces where encrypted checksums or digital signatures are used to establish the authenticity of received information.

In workstation environments the disk-based configurations are well-suited for distributing the processing of sensitive information, especially when the workstation is part of a network. The highly reliable security is applicable to both Government and commercial message and data handling systems to counter many of the security vulnerabilities of contemporary small computers. At the same time, the fact that the iAPX286 was designed for upward compatibility with the popular 8086 and 8088 processors used in the IBM PC and similar machines offers substantial opportunities to adapt existing application software. In addition the high-performance available from multiprocessor configurations offer unique opportunities for specialized systems such as concurrent LISP implementations for decision support and other artifical intelligence application or high-capacity multiuser data base management systems.

In short, the Gemini family of high-performance, secure computer products is a significant step towards the easy availability of useful trusted computer systems for Government and industry.

REFERENCES

1. S.R. Ames, M. Gasser, amd R.R. Schell, "An Introduction to the Principles of Security Kernel Design and Implementation," Computer, Vol. 16, No. 7, July 1983, pp. 14-22.

2. DoD Trusted Computer System Evaluation Criteria, CSC-STD-001-83, 15 August 1983, DoD Computer Security Center, Ft. Meade, Md.

3. 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.

4. R.E. Childs, et. al., "Processor Family for Personal Computers", Proceedings of the IEEE, Vol. 72, No. 3, March 1984, pp. 363-376.

5. 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.

6. J. Rushby and B. Randell, "A Distributed Secure System," Computer, Vol. 16, No. 7, July 1983, pp. 55-67.

7. D.P. Reed, and R.K. Kanodia, "Synchronization with Event-counts and Sequencers," Communications of the ACM, Vol. 22, No. 2, February 1979, pp. 115-124.

8. R. R. Schell, "A Security Kernel for a Multiprocessor Microcomputer," Computer, Vol. 16, July 1983, pp. 47-53.

9. D.L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," Communication of the ACM, Vol. 15, No. 12, December 1972, pp. 1053-1058.

10. M.D. Schroeder, et. al., "The Multics Kernel Design Project," Proc. Sixth ACM Symposium on Operating Systems Principles, November 1977, pp. 43-56.

ON THE INABILITY OF AN UNMODIFIED CAPABILITY MACHINE TO ENFORCE THE *-PROPERTY

W. E. Boebert

Honeywell Systems and Research Center

Minneapolis, MN

ABSTRACT

It is shown that, in the absence of additional mechanisms, a machine based on capability addressing is incapable of enforcing a common class of security policies. This circumstance results from an intrinsic property of such machines: the right to exercise access carries with it the right to grant access. This result was, to the author's best knowledge, discovered by the PSOS [1] design team circa 1979, and is summarized here owing to increased interest in capability machines and the enforcement of this particular class of security policies.

UNMODIFIED CAPABILITY MACHINES

A "capability machine" [2] is one in which there exist distinguished objects of the form (Name, Mode), where Name is the name of a storage object such as a segment, and Mode is an access mode such as Read or Write. Capabilities are protected from tampering and are passed between programs in order to provide controlled sharing of storage objects. In order for a program to exercise access to a storage object, the program must possess a capability granting the desired access. "Possess" in this sense means that the program has access to the storage object which contains the capability. An "unmodified capability machine," in the sense used here, is one in which capabilities are the sole mechanism for controlling access by programs to storage objects.

SECURITY POLICIES

The Simple Security Property and the *-Property

The Simple Security Property and the *-Property are components of a commonly encountered class of mandatory security policies. Such policies define restrictions on the flow of information within a computer system. The restrictions are based on attributes associated with programs in execution (often called subjects) and storage objects (often called objects). A simplified version of these properties is presented below; the interested reader is referred to [3] and [4] for more details.

The attribute associated with a subject is its "clearance," a value which expresses the trustworthiness of the user on whose behalf the program is executing. The attribute associated with a storage object is its "classification," a value which expresses the sensitivity of the information it contains.

The Simple Security Property states that Read access is permitted if and only if clearance is greater than or equal to classification. The *-Property states that Write access is permitted if and only if classification is greater than or equal to clearance.

Trojan Horse Attacks

These restrictions are imposed to prevent what are called "Trojan Horse attacks," in which a malicious program abuses the clearance it temporarily possesses (as the consequence of executing on behalf of a trusted user) in order to compromise the information to which that user has a legitimate right of access. In the typical Trojan Horse attack, a program written by a malicious party is made publicly available. An unwitting user invokes the program, bestowing upon it (for the period of the invocation) the clearance level of the user. The program then performs, in addition to its publicly known function, a clandestine examination and transfer of information to which its author does not have legitimate access.

The appropriateness of the Simple Security Property is obvious, since it states that no program may access information whose sensitivity exceeds the trustworthiness of the user on whose behalf the program is executing. The *-Property is less obvious; it exists to prevent the trivial circumvention of the Simple Security Property by means of Write access. Without the *-Property, it would be possible for a malicious program to write sensitive information (to which it temporarily has legitimate access, as a consequence of being invoked unwittingly by a trustworthy user) into a storage object of low classification. Such information could then be read later by a program executing on behalf of a user of low trustworthiness, thereby compromising the sensitive information. The restriction imposed by the *-Property prevents this "de facto declassification."

POLICY ENFORCEMENT ON AN UNMODIFIED CAPABILITY MACHINE

Consider an omniscient oracle which executes on a pure capability machine for the purpose of enforcing a security policy consisting of the Simple Security Property and the *-Property. Any program wishing access to a storage object must appeal to the oracle, which compares the clearance of the program to the classification of the storage object and grants or denies access accordingly, by setting the Mode value of the capability which is returned as the response to the appeal. Thus a program executing on behalf of a user with a high clearance which requests access to a storage object of low classification would receive a capability with only the Read access set, in accordance with the *-Property.

SUBVERSION OF THE ENFORCEMENT MECHANISM

In a pure capability machine, the intentions of such an oracle can be subverted by the following attack: A malicious program executing on behalf of a user with a low clearance requests a capability which grants Read and Write access to a storage object of equally low classification. We will call this object low_object and the capability RW_low_object. Such a request is naturally granted by the oracle. The program places RW_low_object in low_object. At some later time, a user with a high clearance unwittingly invokes a Trojan Horse program. The Trojan Horse program requests a capability

granting read access to a storage object of high classification. We will call the object high_object and the capability R_high_object. This request will also naturally be granted. Finally, the Trojan Horse program requests a capability granting Read access to low_object. This request will be granted by the oracle, in accordance with the Simple Security Property. We will call this last capability R_low_object.

The Trojan Horse program then uses R_low_object to fetch RW_low_object from low_object. A malicious program now simultaneously possesses R_high_object and RW_low_object, and is therefore able to transfer information in violation of the *_Property.

Note that this attack succeeds even if the *-Property is further restricted to state that writing can only occur if clearance equals classification. The attack can be stopped only if both reading and writing are restricted to cases where clearance equals classification, which is of course the trivial case of no flow whatever.

CONCLUDING OBSERVATIONS

The attack is made possible by an inherent attribute of pure capability machines: the right to exercise access carries with it the right to propagate that access. Thus even if an omniscient oracle correctly creates capabilities, it cannot control their further propagation. If extra mechanisms are imposed to impose this control, the machine is no longer an unmodified capability machine.

REFERENCES

[1] Neuman, P.G., et al, A Provably Secure Operating System: The System, Its Applications, and Proofs, Computer Science Laboratory Report CSL-116, SRI International, Menlo Park, CA, 7 May 1980.

[2] Dennis, J.B. and Van Horn, E.C., "Programming Semantics for Multiprogrammed Computations," CACM, IX, 3, March 1966, pp. 143-155.

[3] Landwehr, C.E., "Formal Models for Computer Security," ACM Comp. Surv. XIII, 3, Sept. 1981, pp. 247-248.

[4] Department of Defense Trusted Computer System Evaluation Criteria, Department of Defense Computer Security Center, 15 Aug. 1983.

A Trusted Computing Base for Embedded Systems

John Rushby

Computer Science Laboratory
SRI International

Abstract

The structure of many secure systems has been based on the idea of a security kernel – an operating system nucleus that performs all trusted functions. The difficulty with this approach is that the security kernel tends to be rather large, complex, and unstructured.

This paper proposes an alternative structure for secure embedded systems. The structure comprises three layers. At the bottom is a *Domain Separation Mechanism* which is responsible for maintaining isolated "domains" (also known as "processes" or "virtual machines") and for providing controlled channels for their intercommunication. The other resources of the system (for example, devices and the more abstract entities, such as file systems, built upon them) are each controlled by independent *resource managers* which comprise the second layer of the system. The *applications* code provides the third layer. Components in both the resource management and applications layers are protected from each other by the domain separation mechanism. The Trusted Computing Base is composed of the domain separation mechanism and a reference validation mechanism associated with each resource.

The benefit of this approach is that it leads to a separation of concerns: each component of the embedded system performs a single, well-defined activity and can be understood (and verified) in relative isolation from all other components. Implementation and language issues are also discussed.

1. Introduction

This paper is concerned with the design of secure computer systems. The DoD Computer Security Center has established criteria which such systems should satisfy and has structured these criteria into several divisions according to the degree of assurance of security that they confer [6]. Although some of the criteria are quite specific, the document in which they are described (the "Orange Book" [6]) falls (intentionally) short of being a "how to" manual on secure systems design. Furthermore, although the document includes a section on the rationale underlying the criteria, many readers continue to find their motivation obscure and their interpretation difficult. In this paper, I will present my understanding of the motivation behind some of the criteria (specifically, those applying to evaluation classes B3 and A1) and I will provide my interpretation of how they should influence the design of secure embedded systems – by which I mean systems dedicated to the support of a single application (I wish to exclude the more complex case of general purpose systems, although many of my observations will apply to that class of systems as well).

It is important to stress that these are my personal opinions and interpretations concerning the DoD Evaluation Criteria; they have not yet been presented to, much less sanctioned by, the DoD Computer Security Center.

2. What's in a TCB?

There is considerable experimental evidence that conventional computer systems are not secure. Furthermore, repairing security flaws as they are discovered has proved an inadequate approach to the provision of truly secure systems. Firstly, many of the flaws in conventional systems reflect fundamental inadequacies in their design and their complete repair is infeasible. Secondly, there is no technique for demonstrating that all the security flaws have been eliminated from a system not designed with that aim in mind; penetration testing only reveals the presence of flaws, not their absence. The only sound approach to the provision of secure computer systems is to design security into those systems right from the start. Furthermore, the primary evidence that a system is secure must be based on the analysis of its design and implementation. The Evaluation Criteria express this as follows:

"Systems representative of higher classes in division B and division A derive their security attributes from their design and implementation structure. Assurance that the required features are operative, correct, and tamper proof under all circumstances is gained through progressively more rigorous analysis during the design process" [6, p5].

Thus, a secure computer system must contain mechanisms that are sufficient to guarantee its security in all circumstances, and it must be possible to provide compelling evidence that those mechanisms are entirely adequate to their task; it is not enough for the mechanisms to be be correct, they must be seen to be so. Generally speaking, small, simple, and localized mechanisms are easier to get correct, and more easily shown to be correct, than large, complex, or diffuse ones. The first task in the design of a secure system, therefore, is to find a way of structuring it so that its security mechanisms are localized as much as possible, and are as small and as simple as possible. In addition, since the evidence for the security of a system is to be provided by analysis of its security mechanisms, those mechanisms must be based on some overall concept of what constitutes security and the evaluation of those mechanisms must be performed with reference to that concept.

The Evaluation Criteria refer to the totality of security mechanisms within a secure system as its *Trusted Computing Base* (TCB). For Evaluation Class B3 and above, it is required that

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

"... The TCB modules shall be designed such that the principle of least privilege is enforced.

"... The TCB shall be designed and structured to use a complete, conceptually simple protection mechanism with precisely defined semantics.[1] This mechanism shall play a

[1]This is an incorrect use of the word "semantics" (which refers to the association of meaning with language). The word "behavior" seems to be intended.

central role in enforcing[2] the internal structuring of the TCB and the system. The TCB shall incorporate significant use of layering, abstraction, and data hiding. Significant system engineering shall be directed toward minimizing the complexity of the TCB and excluding from the TCB modules that are not protection-critical" [6, p37].

The "complete, conceptually simple protection mechanism" endorsed by the Evaluation Criteria is that of a *Reference Validation Mechanism* (RVM), which is the name given to a mechanism that implements the concept of a *reference monitor*. Unfortunately, the definitions given for these notions in the Evaluation Criteria are not particularly helpful. We are told that a reference monitor is

> "an access control concept that refers to an abstract machine that mediates all accesses by subjects to objects" [6, p112],

and that a reference validation mechanism

> "validates each reference to data or programs by any user (program) against a list of authorized types of reference for that user" [6, p64].

In order to understand what is intended by these terms, an analogy with the human world will be helpful. Imagine a bureaucracy that uses untrusted clerks to process classified information. Each clerk is assigned a *level* which is the most highly classified information that he may process: for example, a Secret level clerk may process information classified as Secret, Confidential, or Unclassified. The clerks are assigned to offices according to their levels: all the Unclassified clerks occupy one office, the Confidential ones another, and so on. The information processed by the clerks is recorded in folders stored in vaults. There is a separate vault for each security classification: one contains only Unclassified folders, another contains the Confidential folders etc. Periodically, a clerk will need to retrieve a folder from a vault, or to return one. The exits from the clerks' offices all lead into a central hallway containing the vaults and patrolled by a guard. A clerk who leaves his office in order to retrieve information

---

[2]This also seems an inappropriate choice of words. Structure is a property of a design and cannot be "enforced" by a mechanism which is an artifact of that same design. A better choice of words would be "influencing" or "determining".

from a vault will be intercepted by the guard and allowed only to enter a vault with a classification less than or equal to his own level. Once he has obtained the information required, the clerk will leave the vault and the guard will escort him back to his office. A clerk who wishes to deposit information in a vault will be treated similarly, except that the guard will allow him to enter only a vault with classification *greater than* or equal to his own level. The clerks have no memory of their own, everything they process must be written down on paper; the guard ensures that clerks are empty-handed when they enter a vault classified below their own level, and also when they leave a vault classified above their own level.

It seems clear that this arrangement provides security in the sense that no information derived from a folder in a highly classified vault can ever wind up in a folder stored in a vault of lower classification. The interesting question here is to enquire what it is about the arrangement that makes it secure.

Clearly, the guard plays an important role in the security of this system – he is, in fact, its reference validation mechanism in that he "validates each reference to data (i.e. folders) by any program (i.e. clerk) against the type of reference authorized for that program" [6, p64 paraphrased]. But is the guard the only characteristic of this system that is necessary to its security? Clearly not – certain properties of the *environment* in which he operates are crucial to his ability to perform his task correctly. Suppose, for example, that the different offices were not isolated from one another, but had interconnecting doors. The system would then provide no security at all since Unclassified clerks could enter the Secret office and could there observe and record information classified as Secret. Similar problems would arise if the vaults had interconnecting doors, or if clerks could slip by the guard and evade his supervision. The Evaluation Criteria address these problems by citing the following three design requirements which must be met by a RVM [6, p64].

- The RVM must be tamper-proof,
- The RVM must *always* be invoked, and
- The RVM must be small enough to be subject to analysis and tests, the completeness of which can be assured.

These requirements are often referred to as the *isolation*, *completeness* and *correctness* of an RVM, respectively.

Of these three requirements, only the last (correctness) is really a property of the RVM itself; the other two are more properly regarded as properties of the environment in which the RVM operates. Thus, although the concept of a reference monitor may serve to guide and motivate the design of a TCB, it is clear that a TCB must be more than just an RVM. The *first* requirement of a TCB is that it should create an environment in which an RVM can operate securely.

The type of environment required for this purpose is one of cleanly separated "domains" in which untrusted programs can operate with no opportunity to interfere with each other. Domains cannot be completely isolated, of course, or there would be no possibility for information flow between the different security levels at all – the purpose of a secure system is not to *prohibit* information flow between different security levels, but to *control* it. The Domain Separation Mechanism (DSM) of a TCB must therefore provide channels for inter-domain communication, but these communication channels must be under strict control and must be "wired up" correctly. (For example, there must be no channels directly connecting untrusted domains of different security levels.) Abstract domains and communication channels are represented by the isolated "offices" and "vaults" and by the "doors" and hallways of the human-world example described earlier.

It is interesting to observe that for certain simple systems, domain separation and controlled "wiring" of the inter-domain communication channels may be all that is necessary to provide security: an explicit RVM is not always necessary. In the example we have been considering, it is conceivable that a cunning layout of corridors may make it possible to dispense with the guard altogether – a clerk wishing to obtain information from a vault would leave his office and find himself in a corridor giving access only to the vaults that he is allowed to enter. There is an apparent difficulty here in that in that each clerk is restricted to a *different* set of vaults depending on whether he wishes to read, or to write information. This problem can be overcome by providing different channels for different operations: a Secret clerk wishing to read information from a vault would exit his office by a door which gave him access to the Unclassified, Confidential, and Secret vaults, while one wishing to deposit information would leave by a door leading to the Secret and Top Secret vaults.

We should now ask whether this scheme of carefully routed corridors is really any different to the original one involving the guard. I think not: both schemes are implementations of the reference monitor concept, but whereas the guard is a *run-time* mechanism – checking each access as it is about to occur – the corridor routing approach is applied at *system configuration* time.

So, to summarize so far: domain separation is a necessary prerequisite for the implementation of an RVM. In some simple cases, the RVM can be "hard-wired" into the routing of the inter-domain communication channels. (Certain communications processing applications lend themselves to this approach – see [3, 10] for an example of this type.) In more complex cases, it may be better to use a run-time RVM to check each inter-domain communication as it occurs.

The question we should now consider is whether a run-time RVM should be part of the DSM, or a distinct mechanism in its own right. In the former case, the DSM could monitor each inter-domain communication channel and consult its in-built RVM before allowing the communication to proceed. In the latter case, untrusted domains would not be allowed to directly communicate with each other at all; instead, they would have to relay their communication through a special domain containing an RVM. This reference validation domain would check each communication and pass on only those that were authorized.

On the surface, the second scheme appears to correspond to a better separation of concerns, but would also seem inefficient (in that it doubles the number of communication steps and introduces additional domain swaps). Further consideration, however, reveals that these are not the only two alternatives: there is a third which has certain advantages over both the others. In order to see this, it is necessary to consider the nature of inter-domain communication more carefully.

There are basically two reasons why one domain should need to communicate with another. The first reason is simply for the purpose of passing information: one domain passes information to another that has need of it. This style of communication is common in communications processing applications where messages go through several stages of processing. The second reason is quite different: it is performed in order to obtain a *service*

from a domain which encapsulates a *resource*. Examples of such services include storing and retrieving files, and providing access to a communications line. Different kinds of resource naturally provide different kinds of services and the access control restrictions necessary to enforce security will naturally differ with the different services. For example, it is necessary to prevent domains from reading files classified *above* themselves, and to prevent them from writing files classified *below* their own clearance: the reading and writing of information have different security implications and this is reflected in the different access control restrictions that apply to the two operations. For this reason, it seems reasonable to have a separate RVM associated with each different type of resource; in this way, each RVM can be tailored to the particular characteristics of the service provided by the resource with which it is associated.

A counter-argument maintains that all operations on all resources can be characterized as either read-like or write-like. If this is the case, then only a single RVM is needed: it could operate by first consulting a record of whether the requested operation is read-like or write-like and then applying whichever of the two fundamental access control disciplines is appropriate. The benefit that can be claimed for this approach is that the crucial reference validation function is localized in a single place. This argument seems plausible, but it loses much of its force if resource managers (i.e. the domains which encapsulate resources) have to be *trusted*. For certain simple kinds of resource, it is possible to construct resource managers that do not need to be trusted, but this is possible only if a separate instance of the resource can be synthesized a for each security level, and if the resource manager does not retain private state information. (The vaults in our example have these characteristics.)

In order to understand why it is necessary to synthesize separate instances of a resource at different security levels it will be helpful to return to the analogy with secure office procedures. Suppose that the only communication between our bureaucracy and the outside world is via a single telephone line. When the phone rings, an operator answers it and interrogates the caller in order to determine his identity and authorized security level – this could require that the caller gives a secret password, for example. The telephone operator will then switch the call over the internal telephone system to an office containing clerks of an appropriate level. It is clear that the telephone operator must be trusted to perform these tasks correctly. Of course, we could require that the guard who controls access to the vaults doubles up as the telephone operator, but this evades the issue. The procedures necessary to operate the telephone securely

are quite different to those concerned with access to the vaults and the principle of separation of concerns indicates that they are best treated separately: the best structured mechanism is that which uses a separate and trusted telephone operator (who personifies a resource manager for the telephone line).

To understand the significance of retaining state information in a resource manager, suppose that the vaults do not merely contain a haphazard collection of folders which ordinary clerks can sort through at will, but are instead maintained as orderly filing systems. This could be achieved by having a filing clerk inside each vault who obtains files on behalf of the ordinary clerks and who stores returned files back in their proper places. In order to maintain the filing system, each filing clerk is allowed to maintain a directory recording which file is stored where inside his vault. Now suppose the Unclassified vault contains, among others, 26 folders of different sizes and suppose further that one of the ordinary Secret level clerks makes 6 trips to the Unclassified vault and retrieves the 5th, 14th, 9th, 7th, 13th and 1st smallest folders, in that order. It is not hard to see that the Secret string "ENIGMA" has been communicated to the Unclassified filing clerk – who can now manufacture a folder containing this information and hand it to the next ordinary Unclassified clerk who visits his vault.

These examples should convince the reader that resource managers, such as the telephone operator and the filing clerks, generally need to be trusted to perform their functions *securely* as well as correctly. This being the case, it is surely most appropriate for each resource to have its own RVM associated with it – that RVM can then be designed to integrate cleanly with the other trusted functions of the resource manager.

Summarizing this discussion, I propose that trusted embedded computer systems should be structured as follows. At the bottom, closest to the hardware, there should be a *domain separation mechanism* whose purpose is to divide the system into a number of separate execution domains – virtual machines, in effect – which are interconnected by carefully "wired" communications channels. It is an engineering decision whether the reference monitor function at this level is accomplished by the fixed routing of the the inter-domain communications channels, or by a run-time reference validation mechanism within the domain separation mechanism.

Above the domain separation layer should come a *resource management layer.* Some resource managers will consist of simply the software needed to encapsulate and control some hardware device (i.e. a device driver). Others will synthesize more sophisticated resources out of primitive ones – for example, a file system may be built on top of a primitive disk resource. The code that manages each resource should be isolated in one or more domains; any trusted code must reside in a separate domain from that which is untrusted. Care and skill are needed to minimize the amount and complexity of the trusted code in each resource manager. This can often be achieved by careful layering. For example, it is a complex task to synthesize a secure file system directly on top of a raw disk driver. It may be better to first synthesize securely partitioned "mini-disks" on top of the single physical disk and to then build the secure file system on top of the secure mini-disks.

Access to all "multilevel" resources must be controlled by a reference validation mechanism that provides the only outside interface to the services of the resource. Most often, the reference validation mechanism will be part of a domain that performs other trusted functions concerned with the management of the resource.

At the top, above the resource management layer, should come the *application* layer comprising the code necessary to tailor the system to the intended application. The domain separation provided by the domain separation mechanism must be exploited to separate trusted from untrusted applications code, and to partition untrusted applications code operating with different security attributes (e.g. different "levels").

Within this structure, the Trusted Computing Base consists of the domain separation mechanism, together with all domains that perform trusted functions. These will include the reference validation and other trusted domains from the resource management layer, together with the few (if any) application-specific trusted domains from the applications layer.

This approach to system structuring is very similar to that employed in modern operating systems (e.g. Thoth [5] and Tunis [9]) and is in contrast to the older approach to secure system design in which nearly all trusted functions were combined with the domain separation mechanism to yield a rather large "security kernel" with little internal structure [1, 10]. The approach advocated here extends naturally to distributed systems (where domain isolation is

achieved using separate processors and inter-domain communication uses external communications lines) and seems well able to satisfy the system architecture requirements of the Evaluation Criteria for B3 systems and beyond.

3. Implementation and Language Issues

In order to implement a TCB with the structure described in the previous section, it is sensible to begin with the DSM. One approach is to use *physical* domain separation – that is, to use separate processors for each domain. (See [4, 12] for systems based on this approach.) If, however, a single processor is to support multiple domains, then we have a choice of compile-time or run-time separation mechanisms.

The compile-time approach relies on a programing language implementation to provide separation. In order to be suitable for this purpose, the chosen programming language must be oriented towards the construction of programs from inter-communicating, but otherwise isolated, units (such as "tasks" or "processes"). Modern message-based programming languages for distributed systems (such as CSP and Gypsy) have this character; languages based on shared-variable parallelism are not suitable. The problem with the language-based, compile-time approach to domain separation is that it depends on the correctness of a (generally large and complex) compiler. Although the semantics of the language may indicate that domains with no explicit communication between them are unable to influence each other, there is a danger that a compiler bug, or possibly a deliberately planted Trojan Horse, may permit communication anyway. There is evidence that some real compilers do contain serious security flaws: one notorious example concerns a Fortran compiler that allowed the creation and execution of arbitrary machine code, while a particularly insidious Trojan Horse is described in [13]. Since there is no practical technology for verifying compiler correctness, the Evaluation Criteria appear not to sanction this approach.

Instead, the Evaluation Criteria for Divisions B and A require use of a run-time domain separation mechanism. (The full requirements are introduced at the B2 level, but some of the key requirements are present at the B1 level also.) A run-time domain separation mechanism relies on hardware protection mechanisms to provide domain isolation. The protection mechanisms provided by currently available hardware are usually based on multiple CPU states (at least a superviser/user mode distinction is necessary) together with memory

management functions. These may range from the fairly crude (e.g. the fixed set of PAR/PDR registers provided by a PDP-11/34) to the relatively sophisticated (e.g. the descriptor-based addressing scheme of the Intel iAPX286). The Evaluation Criteria for Class B2 and above require hardware protection mechanisms such as these to be present and to be exploited as follows.

> "The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g. by modification of its code or data structures). The TCB shall maintain process isolation[3] through the provision of distinct address spaces under its control. It shall make effective use of available hardware to separate those elements that are protection-critical from those that are not." [6, p30]

The TCB structure that I am advocating allocates exactly these tasks to its domain separation mechanism. I will use the term *separation kernel* to refer to a run-time DSM which operates in the way described above. In contrast to a conventional "security kernel", a separation kernel does nothing but provide domain separation and this gives it a conceptual simplicity that is lacking in a security kernel. The next step is make the implementation correspondingly simple.

The task of a separation kernel is to manipulate the protection features of the hardware in order to manufacture separate domains. The kernel must also provide inter-domain communication and synchronization mechanisms and should probably hide some of the less attractive hardware features (such as interrupts – these should be mapped onto the standard inter-domain communication mechanism). By removing all complex tasks from the separation kernel, each of its remaining functions can be accomplished in a small fixed number of machine instructions. This makes it possible for the kernel to run with interrupts masked off. This enormously simplifies the kernel and contributes greatly to its comprehensibility, since it can now be understood as a single sequential program.

The functionality of a separation kernel as described here is almost identical to that of the "nucleus" of any modern operating system [5, 9] and is therefore based on a well understood

---

[3] I interpret "process isolation", in the sense used here, as synonymous with domain separation.

technology. The primary difference between a conventional nucleus and a separation kernel is that the latter must provide absolutely rigorous separation between its client domains. Considerable simplification is possible in the case of embedded systems since their process structure is generally static. The separation kernel for an embedded system need not, therefore, support dynamic domain creation, and can use very simple domain scheduling strategies. Such a separation kernel should require no more that a *couple of hundred* machine instructions in total.

An important question concerns the choice of language in which a separation kernel should be programmed. Most high-level programming languages are ruled out immediately since they depend upon a run-time support system that is quite often *larger* than the kernel we wish to construct. Since the separation kernel is to be the fundamental security mechanism in the system, its behavior cannot be allowed to depend on any code but its own. Thus, the kernel implementation language must not require a run-time support system, it must permit special hardware registers to be named directly, and it must allow the use of special machine instructions. These requirement rule out all but assembler and a few system implementation languages, such as the sequential subset of Concurrent Euclid [9]. Personally, I can see no reason for using anything other than assembler since the code of a separation kernel consists almost exclusively of assignments to special hardware registers (e.g. loading the protection status or the memory management registers) and special instructions (e.g. those to set the interrupt mask or the processor status word). The behavior of a separation kernel written in a high level language cannot be inferred from the standard semantics for that language; its behavior and effects are primarily determined by the characteristics of the *hardware* which it manipulates.[4] For this reason, it is absurd to suggest that a separation kernel should be written in a high-level language in order that it be verifiable. The effect ascribed to the assignment statement `x := 0` by a standard programming language semantics does not begin to address the real behavior of the program when `x` is the processor status word! The task of verifying a separation kernel is rather different than conventional program verification. A technique for accomplishing the task is described informally in [10] and more formally in [11]. An application of the technique is described in [8].

---

[4] A simple separation kernel written in the sequential subset of Concurrent Euclid is described in [7]. However, I found the assembler code from which it was derived considerably easier to comprehend.

Although a separation kernel must essentially be written in assembler (or in assembler disguised as a high-level language), one of the merits of the TCB structure advocated here is that it isolates all the machine dependencies in the (very small) separation kernel so that the rest of the system *can* be written in a high-level language. So the next question concerns the choice of language to be used in the rest of the system – that is, in the resource management and application layers. Essentially, any language, or set of languages, *could* be used, since each domain may contain whatever run-time support mechanisms are necessary for languages used within that domain. Inter-domain communication can be provided by subroutine calls on the separation kernel interface.

But while it may be *possible* to use any language(s) whatever, it does not follow that all languages are equally appropriate. An embedded system presumably has some overall purpose: all its domains must cooperate towards that end and must be understood in combination with each other. The programming language used should therefore be one which is matched to the environment created by a separation kernel: that is one of isolated domains with controlled inter-domain communications channels. Essentially, this environment simulates that of a *distributed system* and so the most appropriate programming languages are those intended for distributed systems. If such a language is used, then the separation kernel becomes, in effect, part of its run-time support system and the inter-domain communication facilities provided by the kernel must correspond to those assumed by the language. Unfortunately, the design of programming languages for distributed systems is still in its early stages and for systems to be constructed in the near future, it will probably be necessary to graft inter-domain communication primitives onto an existing sequential programming language whose choice is dictated by other factors.

Design of the inter-domain communications primitives that should be supported by the separation kernel is an interesting problem (see [2] for a discussion of communications primitives). Recall that the resource management layer of the proposed TCB structure provides *services* to *callers*. The form of communication mechanism that is most appropriate to this form of interaction is that of a *Remote Procedure Call* (RPC): the domain requesting the service must call the domain that provides (the interface to) it and must wait until the requested service has been performed and its results (if any) returned. From the caller's point of view, the behavior of an RPC is identical to that of an ordinary procedure call: the fact that

service is actually provided by a remote domain is invisible. As well as a simple semantics, RPCs have a simple message-passing implementation using a "blocking send with reply" and a "blocking receive" [2, 5].

In contrast to the resource management layer, where RPCs provide the communication mechanism of choice, inter-domain communication in the application layer may be better served by straightforward message passing (i.e. "non-blocking" send and receive). However, the implementation of non-blocking send and receive is more complex than that of their blocking counterparts, since it requires the buffering of messages that have been sent but not yet received. To my mind, the implementation of these primitives is more complex than is desirable within a separation kernel (remember, we want each kernel operation to execute in a fixed, small number of machine instructions, so that it will be safe to mask interrupts during the interval). A compromise approach is possible, however, by providing non-blocking message passing as a service of the resource management layer. This can be accomplished using a "queue management" domain that responds to RPCs containing requests to enqueue a message from one domain to another, or to dequeue a message from the input queue of its caller.[5]

As well as requiring only a simple, relatively efficient implementation, the use of RPCs as the basic inter-domain communication mechanism has another advantage: it can support the use of Ada. It is likely that many future secure system developments will require use of Ada. Unfortunately, Ada was designed while hardware and language technology were in transition [14] and its tasking facilities contain shared-variable features that are poorly matched to the environment of a distributed system – and to the environment described here. However, the Ada rendezvous is essentially an RPC mechanism [2] and my suggestion for accommodating Ada within secure systems development is to use the rendezvous as the (only) inter-domain communication mechanism. Within each domain, full Ada will be made available

---

[5]The question of how the messages constituting an RPC call and its reply are actually moved from one domain to another is an interesting one. The conventional solution of simply passing a pointer into a global message pool is obviously unsecure. Modern hardware often provides mechanisms whereby a segment containing a message can be mapped out of one domain's address space and into another, but it is not easy to integrate this mechanism into a high-level programming language. Also, the sanitization of message buffers required by the Evaluation Criteria [6, p15] renders this solution less attractive (in the absence of "write-only" protection). Physical copying of messages from one domain to another seems the most practicable mechanism.

(including unrestricted multi-tasking and shared-variable communication) by an Ada run-time support system contained within the domain. Between tasks located in different domains, however, communications will be allowed only via the rendezvous mechanism, which in this case will be provided by the separation kernel rather than the standard run-time support. The integration of the inter-task rendezvous provided by the separation kernel and the intra-domain communications provided by the Ada run-time support system should eventually be made as seamless as possible – though it may require considerable research and development to reach this stage.

4. Summary and Conclusion

I have described a system structure suitable for implementing secure embedded systems. The structure is comprised of three layers. At the bottom is a *Domain Separation Mechanism* which is responsible for maintaining isolated "domains" (also known as "processes" or "virtual machines") and for providing controlled channels for their intercommunication. The other resources of the system (for example, devices and the more abstract entities, such as file systems, built upon them) are each controlled by independent *resource managers* which comprise the second layer of the system. The *applications* code provides the third layer. Components in both the resource management and applications layers are protected from each other by the domain separation mechanism. The Trusted Computing Base is composed of the domain separation mechanism and a reference monitor associated with each resource. This approach to system structuring is very similar to that employed in modern operating systems (e.g. Thoth [5] and Tunis [9]) and is in contrast to the older approach to secure system design in which nearly all trusted functions were combined with the domain separation mechanism to yield a rather large "security kernel" with little internal structure [1, 10].

The implementation of a domain separation mechanism is called a separation kernel. There is little merit in attempting the implementation of a separation kernel in anything other than assembler; the total size of a separation kernel should be no more than a couple of hundred machine instructions. A message-based implementation of remote procedure calls is the most appropriate choice for the inter-domain communication mechanism to be provided by a separation kernel. More complex communications mechanisms can be provided as services of the resource management layer.

The resource management and application layers can be written in any high level language. It is proposed that Ada can be accommodated by mapping the inter-task rendezvous onto the inter-domain remote procedure call provided by the separation kernel.

The approach advocated here extends naturally to distributed systems (where domain isolation is achieved using separate processors and inter-domain communication uses external communications lines) and seems well able to satisfy the system architecture requirements of the Evaluation Criteria for B3 systems and beyond.

References

1. S.R. Ames Jr., "Security Kernels: a Solution or a Problem?," *Proc. 1981 Symposium on Security and Privacy,* Oakland, CA., pp. 141-150, IEEE Computer Society, April 1981.

2. G.R. Andrews and F.B. Schneider, "Concepts and Notations for COncurrent Programming," *ACM Computing Surveys,* Vol. 15, No. 1, pp. 3-43, March 1983.

3. D.H. Barnes, "The Provision of Security for User Data on Packet Switched Networks," *1983 IEEE Symposium on Security and Privacy,* Oakland, CA., pp. 121-126, IEEE Computer Society, 1983.

4. T.A. Berson, R.J. Feiertag, and R.K. Bauer, "Processor-per-Domain Guard Architecture," *1983 IEEE Symposium on Security and Privacy,* Oakland, CA., pp. 120, IEEE Computer Society, 1983, (Abstract only).

5. D.R. Cheriton, *The Thoth System: Multi-process Structuring and Portability,* North-Holland, Operating and Programming Systems Series, 1982.

6. Department of Defense, Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria,* 1983, CSC-STD-001-83.

7. P.F. Fisher, "An Operating System Security Kernel," Master's thesis, Computing Laboratory, University of Newcastle upon Tyne, England, September 1982.

8. B.A. Hartman, "A Gypsy-Based Kernel," *Proc. 1984 Symposium on Security and Privacy,* Oakland, CA., pp. 219-225, IEEE Computer Society, April 1984.

9. R.C. Holt, *Concurrent Euclid, the UNIX System, and TUNIS,* Addison-Wesley, 1983.

10. J.M. Rushby, "The Design and Verification of Secure Systems," *Proc. ACM 8th Symposium on Operating System Principles,* Asilomar, CA., pp. 12-21, December 1981, (ACM Operating Systems Review, Vol. 15, No. 5).

11. J.M. Rushby, "Proof of Separability - a Verification Technique for a Class of Security Kernels," *Proc. 5th International Symposium on Programming,* Turin, Italy, pp. 352-367, M. Dezani-Cianaglini and U. Montanari, eds., Springer-Verlag Lecture Notes in Computer Science, Vol. 137, April 1982.

12. J.M. Rushby and B. Randell, "A Distributed Secure System," *IEEE Computer,* Vol. 16, No. 7, pp. 55-67, July 1983.

13. K. Thompson, "Reflections on Trusting Trust," *CACM,* Vol. 27, No. 8, pp. 761-763, August 1984.

14. P. Wegner, "Capital-Intensive Software Technology," *IEEE Software,* pp. 7-45, July 1984.

Secure Communications Processor Research

Dr Derek Barnes

Royal Signals and Radar Establishment,
St Andrews Road,
Malvern, Worcestershire,
United Kingdom.

Abstract

For several years, the Royal Signals and Radar Establishment has been carrying out a very active research programme in the area of Secure Communications Processors (SCPs). This work aims to realise a range of practical Trusted Computing Bases (TCBs) for a wide variety of network security applications. The Secure Communications Processors have been designed to be simple, secure and efficient, whilst the underlying TCBs are intended to be application independent. Thus, the cost of developing a new secure component of a distributed system is minimised.

At present, three different Secure Communications Processors have been produced, or are being researched. These are SCP1, a limited functionality TCB for dedicated applications, SCP2, a mid-range TCB for a wide range of network security applications, and SCP3, a high functionality, new architecture, multi-processor system.

In this paper the three current Secure Communications Processors are functionally described, together with the type of applications which they are intended to host. As an example, the role of SCP1 and SCP2 in the practical realisation of the Distributed Secure System is outlined.

1. Introduction

Communications networks and distributed systems exist to provide their authorised users with access to information, and to provide facilities to manipulate that information. The information entrusted to such a system must be valid and reliable, so in many environments it must be protected from unauthorised access and alteration, or at the very least any changes, deliberate or accidental, should be detected. Users should not be able to make use of resources to which they are not entitled; for example, not everyone should be able to make use of system management facilities. Equally important, though perhaps the most difficult to achieve, is to

stop accidental or deliberate attempts to prevent users from using services to which they are entitled. System security is therefore concerned with both accidental and deliberate attacks on the integrity of the system and the information which it contains.

Common-user distributed systems are currently very much in favour for their ability to provide survivable, integrated communications and computing facilities at an economical cost to individual groups of users. However, these groups of users often wish to operate within separate compartments, for example closed user groups, while still communicating via the same common-user system.

It is therefore becoming increasingly important to be able to provide multi-level secure data networks and distributed systems for a wide range of defence and civilian applications.

The Royal Signals and Radar Establishment, part of the United Kingdom Ministry of Defence, has for several years been carrying out a very active research programme in this area. One outcome of this work has been the realisation of a range of Secure Communications Processors (SCPs) to form the basis of various network and distributed system security schemes. Each SCP essentially represents a real-time Trusted Computing Base, specifically orientated towards network security requirements.

2. Trusted Computing Bases

A Trusted Computing Base (TCB) is defined to be the totality of the foundation software and hardware necessary to securely enforce a security policy upon all operations of a computer system.

At its simplest, a TCB consists of a simple Separation Kernel [1], running on comparatively unsophisticated hardware, e.g. a computer with a simple memory management unit. However, a TCB can also be a large Security Kernel, together with all its related non-kernel trusted processes, running on a larger machine and providing general purpose computing facilities [2].

In order to overcome these confusions, we partition the range of possible TCBs into three classes. They are:

The Dedicated TCB

This is the simplest form of TCB; one which is basically dedicated towards performing one very simple, static form of security application. Dedicated TCBs are usually found to be an embedded part of a system component, e.g. providing the foundation for a guard or one-way filter. This type of TCB is preconfigured at system build time, and contains no dynamic process creation facilities. The simple separation kernel [3] represents a typical dedicated TCB.

The Mid-Range TCB

The mid-range TCB provides a common TCB for a related range of applications, e.g. a communications orientated mid-range TCB could form the foundation for a range of network security devices. This type of TCB provides dynamic process creation/deletion facilities but does not allow general on-line user programming. It is therefore particularly appropriate for running fixed applications functions, which are more sophisticated than those which can be supported by a dedicated TCB.

The General Purpose TCB

The general purpose TCB provides the basis for multi-level secure, general purpose, multi-user computer systems, typically allowing its users to perform the general on-line computing functions offered by a large mainframe. KSOS [2] is an example of a system incorporating a general purpose TCB. This type of TCB represents the ultimate in generality for multi-level secure systems.

3. Secure Communications Processors

The RSRE "Multi-Level Secure Communication Networks Initiative" is intended to realise a range of three principle Secure Communications Processors. The essential properties of these three are as follows.

SCP1 – SCP1 is a dedicated TCB for simple, static security applications. The SCP1 work is based upon a generalisation of the SUE Security Kernel work [1] on PDP11/34s carried out at RSRE, but it is intended to realise the SCP1 Separation Kernel concept upon other machines as necessary.

Essentially the SCP1 kernel segments a computer into a preconfigured number of static isolated regimes, or virtual machines. Each regime contains either all trusted or all untrusted functions. The only way in which the regimes can communicate is via preconfigured message passing routes through the trusted SCP1 kernel. The kernel provides scheduling, timing, control and error handling functions, but all input/output operations are directly handled by the appropriate regimes themselves.

SCP2 – SCP2 is a mid-range TCB for network orientated security applications which require dynamic features, but which are removed from the complexities of online user programming. The key features for SCP2 have been to design a system which is secure, yet efficient enough to handle demanding real-time applications, and which has an intrinsically simple design structure.

The philosophy behind SCP2 has been to select very carefully the commercially available computer hardware which best provides the features determined to be desirable for this type of secure application, making absolutely sure that it contains no undesirable features. It should be noted that determining the absence of undesirable features represents an onerous task! The SCP2 work has therefore concentrated upon the design and implementation of the software for a network orientated mid-range TCB, given certain hardware characteristics.

SCP2 provides facilities for dynamic creation and deletion of protected (isolated) process within security environments, and provides controlled message passing inter-process communications facilities. A prototype SCP2 has been designed and built by TSL Communications. The full SCP2 implementation, together with the T-HOST demonstration application, is just commencing with a very short timescale for completion.

SCP3 – SCP3 is a mid-range TCB, similar to SCP2, but offers improved security, integrity and denial of service properties, and is capable of supporting a greater range of applications (including some online user programming).

SCP3 developed from the philosophy that all currently available commercial hardware provides at best, something of a compromise in terms of their features from a security point of view. Thus we have started with no preconceived views, and are designing and implementing a secure system (hardware, firmware and software) to provide a really very secure real-time mid-range TCB for a wide range of network security applications.

When complete, SCP3 will be a modular, secure, multi-processor system, with excellent internal integrity and denial of service protection features.

It is not expected that the SCP3 work will provide a usable secure system for many years. It represents part of our long term research into the next generation of secure systems.

4. SCP Network Security Applications

Having identified the generic TCB types, and looked at the facilities offered by the RSRE range of SCPs, we now consider the type of network security applications for which they can be used.

SCP1 is ideally suited to the provision of simple end-to-end encryption facilities [1]. However, it can be used for a number of guard and filter applications. An example of its use for performing a number of these features is to be found in the Distributed Secure System Project mentioned below.

SCP2 will be able to support a range of applications in a secure, simple and efficient manner for a complete range of security critical components for wide area and local area networks and distributed systems. Applications therefore include the following:

a) Simple Hosts which provide predetermined, limited functionality, computer systems accessed by local or remote users.

b) Network Front Ends which act as security managers between large host computer systems and various networks.

c) Switches. SCP2 can provide the trusted, secure basis of packet switches, message switches, and similar devices, being specifically designed to be efficient enough to handle such demanding real-time applications.

d) Gateways between two or more networks which have different security constraints and require some trusted mediator of the information flowing between them.

e) Network Access Control Centres are required in most local and widely distributed secure systems, providing security management functions.

f) Packet Assemblers/Disassemblers are becoming increasingly security relevant, enabling users to dynamically gain access to multi-level secure facilities and information.

g) General Servers on both local and wide area networks are increasingly being seen as security relevant, e.g. filing systems and electronic mail machines.

SCP3 will be able to support all of those applications outlined for SCP2, as well as more complex ones, including those which require a level of user programming.

The SCP2 implementation will be tested in a demonstration application, known as the T-HOST. This has been very carefully chosen, so that it can form the basis for further research, demonstrates a useful application, and represents a sensible complexity of application implementation. The T-HOST is a simple electronic mail host which can be demonstrated stand-alone, or in conjunction with other T-HOSTs attached to a network. Each T-HOST requires one link to an insecure X25 network, such as the British Telecom PSS network. All inter T-HOST communications are encrypted at the transport service layer. The T-HOST only provides a simple multi-level secure electronic mail application to users on the same, or other T-HOSTS. Users can "login" locally at different security compartments, and once satisfactorily authenticated can use the mail facilities locally or remotely on another T-HOST. A multi-compartment disc backing store is provided on each T-HOST to support the mail application.

5. The Distributed Secure System Project

It is one thing to produce building blocks for multi-level secure systems, and quite another to produce actual working systems. The Distributed Secure System project aims to provide a working demonstration of a real multi-level secure, general purpose distributed system. Rushby and Randell [4] have described the basic concepts of the DSS design. A working non-secure emulation of the DSS design has recently been produced for RSRE by Systems Designers Limited running on a number of DEC PC350 Personal Computers interconnected by a Local Area Network. This emulation has proved the practical feasibility of the basic DSS design.

In the near future, the real DSS implementation work will begin. For this Motorola 68010s will be used for the essential Trustworthy Network Interface Units (TNIUs) which enforce the security separation between the system components. An SCP1 type of separation kernel will provide the dedicated TCB functions required within each TNIU. Initial multi-level secure components such as the Multi-Level Secure Filing System will be very heavily based upon the SCP2 mid-range TCB.

6. Conclusions

The range of Secure Communications Processors currently being pursued by RSRE represent an essential commitment to the basic building blocks for secure computer networks and secure distributed systems. The alternative of starting from scratch with a bare machine for each of the many security relevant applications envisaged has been avoided. This should mean that new, network orientated security applications can be mounted quickly and at minimum incremental cost in the near future.

References

1. D.H. Barnes, "The Provision of End To End Security for User Data on an Experimental Packet Switched Network," *Proc. 4th International Conference on Software Engineering for Telecommunications Switching Systems,* Warwick, England, pp. 144-148, IEE, July 1981.

2. E.J. McCauley and P.J. Drongowski, "KSOS - The Design of a Secure Operating System," *National Computer Conference,* pp. 345-353, AFIPS Conference Proceedings, 1979, Vol. 48.

3. J.M. Rushby, "The Design and Verification of Secure Systems," *Proc. ACM 8th Symposium on Operating System Principles,* Asilomar, CA., pp. 12-21, December 1981, (ACM Operating Systems Review, Vol. 15, No. 5).

4. J.M. Rushby and B. Randell, "A Distributed Secure System," *IEEE Computer,* Vol. 16, No. 7, pp. 55-67, July 1983.

Specifying Multi-Level Security in a Distributed System

Janice I. Glasgow
Glenn H. MacEwen
Takis Mercouris
Farid Ouabdesselam

Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada, K7L 3N6
613-547-2915

Abstract

This paper reports on the status of work to develop a formal model of security for the SNet multi-level secure distributed system. The design of the system is briefly described along with an informal statement of the security model. Two of three parallel efforts to formally specify the top-level of the system are described: a Lucid/dataflow method, and an event-based method. An algebraic specification is also under development but is not described. For each of the two, the formal model is given in the notation of the formalism, and examples of the formal specification are shown. One of the three methods will be chosen for complete specification of the design.

1. Introduction

This paper reports on current work on the development of a formal security model and the formal specification of a previously designed multi-level secure distributed system [MacEwen84a]. The system, called SNet, has been prototyped using Concurrent Euclid on a centralized system. It will shortly be carried to a more advanced prototype stage by moving it to a distributed system based on Ethernet. Further implementations will involve specially designed network interface hardware.

This work is supported by the
Natural Sciences and Engineering Research Council

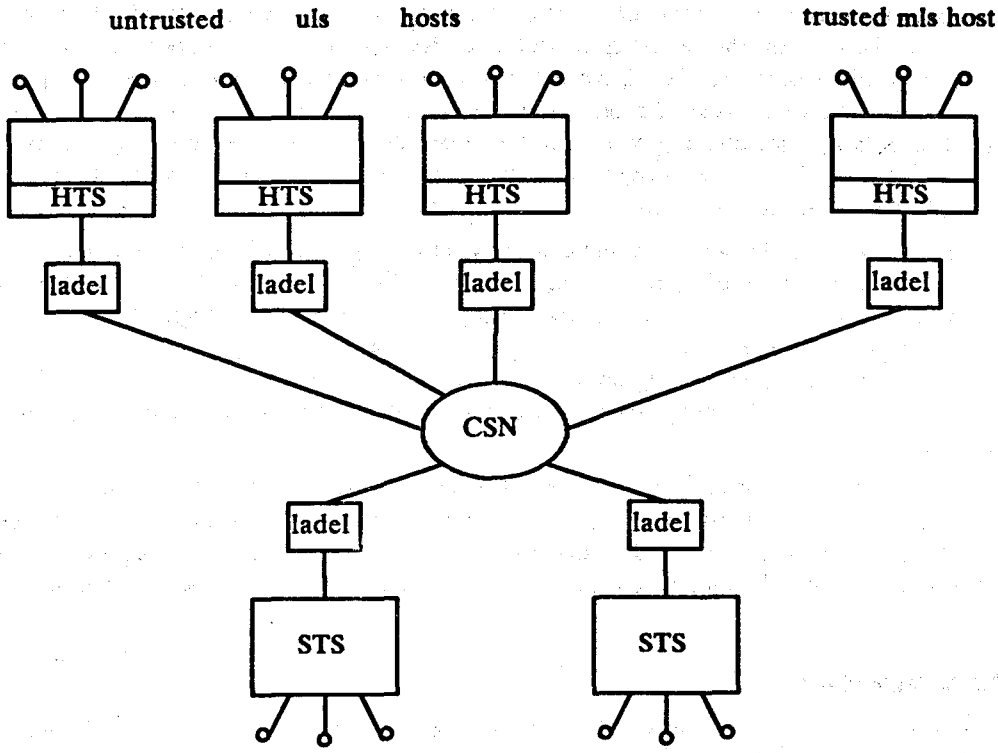


Figure 1. SNet Architecture

Figure 1. shows an overview of the architecture of the design. A communication subnetwork (CSN) links two kinds of hosts: untrusted uni-level secure (uls) hosts, and trusted multi-level secure (mls) hosts. All data on the former are treated with the assumption that they are at a level associated with the host. All data on the latter are associated with a level that may vary up to a maximum associated with the host.

The hosts connect to the CSN via a special interface called a labeller/delabeller (ladel). This interface is a trusted component with three major functions: First, it labels all outgoing messages with the appropriate security level; for uls hosts this is its associated level, for mls hosts the host provides the level. Second, it ensures that messages cannot be tampered with and that a received message originated at a trusted ladel. Third, it enforces multi-level information flow control on incoming messages according to the common lattice security model.

Appearing to the ladels and CSN as mls hosts is a set of trusted components called secure terminal servers (STS's). These components provide a small set of commands for users to establish open circuits to a set of hosts, state the level at which they should be treated for each of these circuits, and to connect to one of these circuits. Otherwise, an STS simply provides a transparent circuit from a user terminal to its currently connected host. Files can be transferred from one host to another by using untrusted protocol software residing on each host and shown in Figure 1. as host terminal-server (HTS) software. Of course, all terminal/host and host/host communication is subject to enforcement by the trusted ladels. One example of a mls host is a secure downgrader.

While the prototyping work is proceeding we are working on a formal security model and the formal specification of the system. Obviously, the prototype development and the specification will interact. The prototype work can expose issues that the specifiers may have missed, and the specification can expose design features that may be incorrect or unnecessary.

We have taken the approach of investigating three different specification methods. Also, the definition of the security model and the investigation of specification methods have proceeded in parallel. We have attempted a top-level specification of the prototype system in each of three formalisms: event-based specification, a dataflow approach based on the Lucid language, and algebraic specification. As a result, the specification work has raised questions about the model and has resulted in a better model that is more specification method independent.

The security model was first defined informally. It's formal statement, of course, will be given in the language of each specification formalism. What we hope to gain from this three-pronged attack is a better understanding of the problems of specifying distributed systems, an improvement in the security model, a better assurance of the correct specification of the system by comparing the three specifications, and finally, a firm base upon which to select the most appropriate specification method to be used for the complete specification.

The following section presents the informal security model. The next two sections each show a formal security model and some examples of the specifications for two of the three specification methods under investigation: the Lucid/dataflow approach and the event-based method. The algebraic specification will be described in a future paper.

2. *Security Model*

The informal security model is based on four kinds of entities: a set U of ***users;*** a set H of ***hosts;*** a set N of ***names,*** N = U ∪ H; and a set L of ***levels.***

A partial order ≤ , called ***dominates,*** is defined on L. Consequently,

$(\forall l)\ (l \leq l)$
$(\forall l1)(\forall l2)(l1 \leq l2 \text{ and } l2 \leq l1 \rightarrow l1 = l2)$
$(\forall l1)(\forall l2)(\forall l3)\ (l1 \leq l2 \text{ and } l2 \leq l3 \rightarrow l1 \leq l3)$

There are three constant mappings:

Clearance: U → L
Max: H → L
Trusted: H → Boolean

Clearance represents the maximum level that a user may specify for his current level. Max represents the maximum level for a mls host and the singular level for a uls host. Trusted indicates whether a host is mls (true) or uls (false).

The unit of user/host and host/host communication is the data type D. Messages of type D comprise the major part of the system state of the model but the model does not explicitly define the complete state since, for a distributed system, the notion of state is not appropriate. However, there is one component of the system state that is security relevant and is, therefore, part of the model. This component comprises two mappings that give, for a user, his currently connected host and associated level.

Currenthost: U → H
Currentlevel: U → L

The functionality of these two mappings is referenced in the following and is denoted by Mh and Ml to mean respectively (U,H) and (U,L).

Users have available the six operations listed below. In these syntactic definitions, Mh and Ml represent the system state information that is security relevant. The definitions are not intended to model the implemented operations directly since the

parameters of type U, and functionality Mh, and Ml can be expected to be supplied by the system. Lower case letters are used to denote an actual parameter of the type specified. The informal semantic description is not part of the model.

usend(U,D,Mh,Ml)
User u transmits a message d with associated level l to destination h, where h=mh(u) and l=ml(u).

ureceive(U,Mh,Ml) returns D
User u receives message d.

login(U,H,L,Mh,Ml) returns Mh x Ml
User u establishes a circuit between himself and host h with his level for that connection being l.

logout(U,H,Mh,Ml) returns Mh x Ml
User u deletes the circuit between himself and host h.

connect(U,H,Mh,Ml) returns Mh x Ml
User u establishes host h as his current host with his level being that previously specified in a login.

disconnect(U,Mh,Ml) returns Mh x Ml
The current host and current level for user u become null.

Hosts can perform two operations.

hsend(H,L,D,N)
Host h transmits a message d with associated level l to destination n.

hreceive(H) returns L x D
Host h receives message d with associated level l.

An ***event*** is a set of associated values. Events are partially ordered on the relation ***precedes***. The model has two types of events,

send(N,L,D,N)
receive(N,L,D)

with values of the types indicated. The execution of certain operations is associated with one and only one event as follows:

usend(u,d,Currenthost,Currentlevel) is associated with an event send(u,l,d,n)
ureceive(u,Currenthost,Currentlevel) is associated with an event receive(u,l,d)
hsend(h,l1,d,n) is associated with an event send(h,l2,d,n)
hreceive(h) is associated with an event receive(h,l,d)

The following four constraints represent the security requirements for the model.

1. Users' levels
For all u $\in$ U, Currentlevel(u) $\leq$ Clearance(u)

Users are constrained to specify a current level only as great as their clearance.

2. Labelling of messages
   For all events send(h,l,d,n), if not Trusted(h) then l = Max(h)
   For all events send(u,l,d,n), l = Currentlevel(u)

All transmitted messages are labelled with a level that is determined by the level of the source. In the case of a mls host, the host must supply this level.

3. Authenticity of messages
   For any set of k identical events receive(n1,l,d), there is a
   set of k distinct preceding events send(ni,l,d,n1), i=1,k.

This constraint covers three concerns. First, all messages received are guaranteed to have originated at a system port, either a user or a host. That is, the network does not generate any messages internally and any message received is guaranteed to have originated at an authentic network source.

Second, any message received must not have been altered with respect to the values of the level, data, and destination address. Consequently, the binding of the level to the data is assured.

Third, duplication of messages is not allowed. This constraint is included to prevent certain information flow channels that might exist if untrusted network components could resend message copies in a modulated stream to encode information that could have been obtained from control information portions of messages. The constraint that the destination component of a message not be altered seems to be over-specification with respect to security concerns, but the duplication restriction seemed to be difficult to express without it.

4. Flow control
   For all events receive(h,l,d), $l \leq$ Max(h)
   For all events receive(u,l,d), $l \leq$ Currentlevel(u)

This is the fundamental multi-level security constraint of "flow upward only" with respect to the dominates relation.

4. Lucid/Dataflow Specification

The Lucid/dataflow specification provides a particularly simple and intuitive model of the system. This is done by utilizing a dataflow network to illustrate the flow of messages through the network. The constraints placed on the flow are represented by the dataflow language Lucid [Ashcroft83]. Because Lucid is also a specification language the end result is a formal specification for the network that can be verified using the Lucid verification rules. Unlike most specifications, the Lucid approach can also provide an implementation for the system.

4.1 Dataflow Model

Dataflow programming can be conceptualized by a dataflow network. Such a network consists of a system of nodes connected by communication arcs. Each node can be considered as a processing module whose only interaction with other modules is through the communication channels.

We can consider a dataflow network for the secure distributed system as consisting of three connected components: users, hosts, and a medium (See Figure 2.).

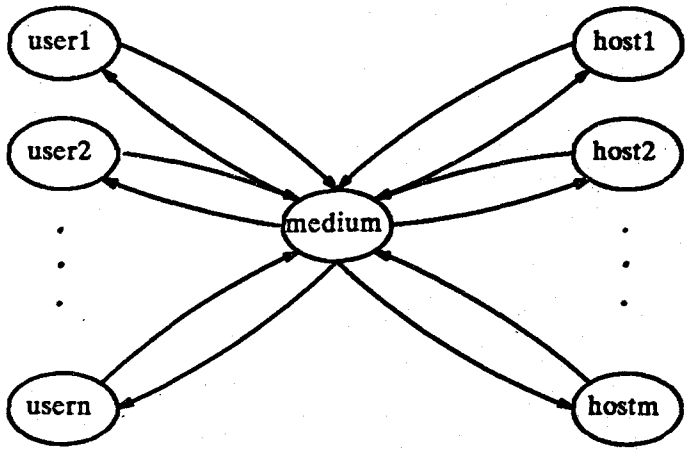


Figure 2. Dataflow Model of SNet

Each user consists of three processing modules (nodes). These nodes represent functions that send messages (usend), receive messages (ureceive) and determine the current host and level (active). The subnetwork for user u is illustrated in Figure 3. The host subnetwork contains, for each host, a receive processing node (hreceive) and a send processing node (hsend) (See Figure 4.). Interconnecting all users and hosts is a medium that receives messages from the hosts and users and transmits messages back to the same set of hosts and users. Figure 5. illustrates the configuration for a two host (h1 and h2) and two user (u1 and u2) distributed network.

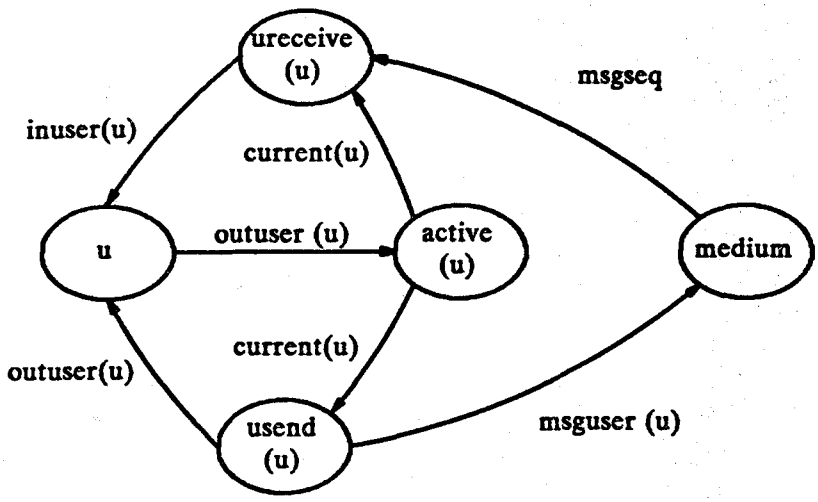


Figure 3. User Dataflow Subnetwork

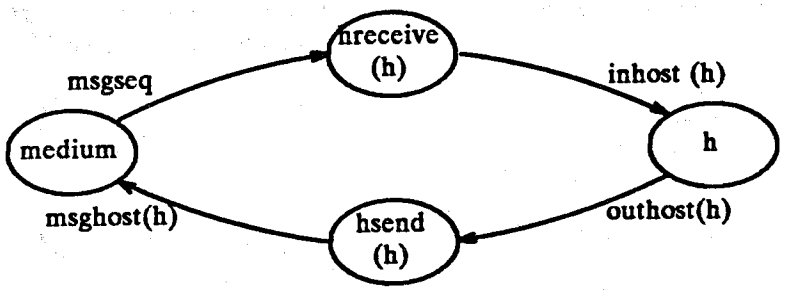


Figure 4. Host Dataflow Subnetwork

4.2 Lucid Specification

The language Lucid was originally developed for the purpose of program verification. More recently, it has been viewed as a special purpose dataflow language. Unlike most other dataflow languages it does not rely on the old von Neumann form of computation. Instead, it intuitively reflects the concepts of data flowing through a network.

Lucid denotes a class of languages. We consider Lucid[A] to be a language that depends on a given algebra A. Operations in a Lucid language consist of pointwise extensions to the operations of the algebra A along with special modal operators. The basic data objects of Lucid[A] are infinite sequences consisting of the data objects of A. These sequences intuitively correspond to the communication channels of a dataflow network. In both cases we are considering infinite history sequences of the form $<p_0,p_1,p_2,...>$ where p_i is said to be the value of sequence p at time i. Our use of time here must be clearly understood; this refers to a local time for the processing node producing the sequence.

Time is used in the Lucid specification to establish a partial ordering (to be introduced later) on related events. That is, a receive event occurs at a time j greater than the time i at which the message was sent. We are not concerned with the time interaction of unrelated events such as two sends. Implementations exist to support this kind of specification [Lamport78].

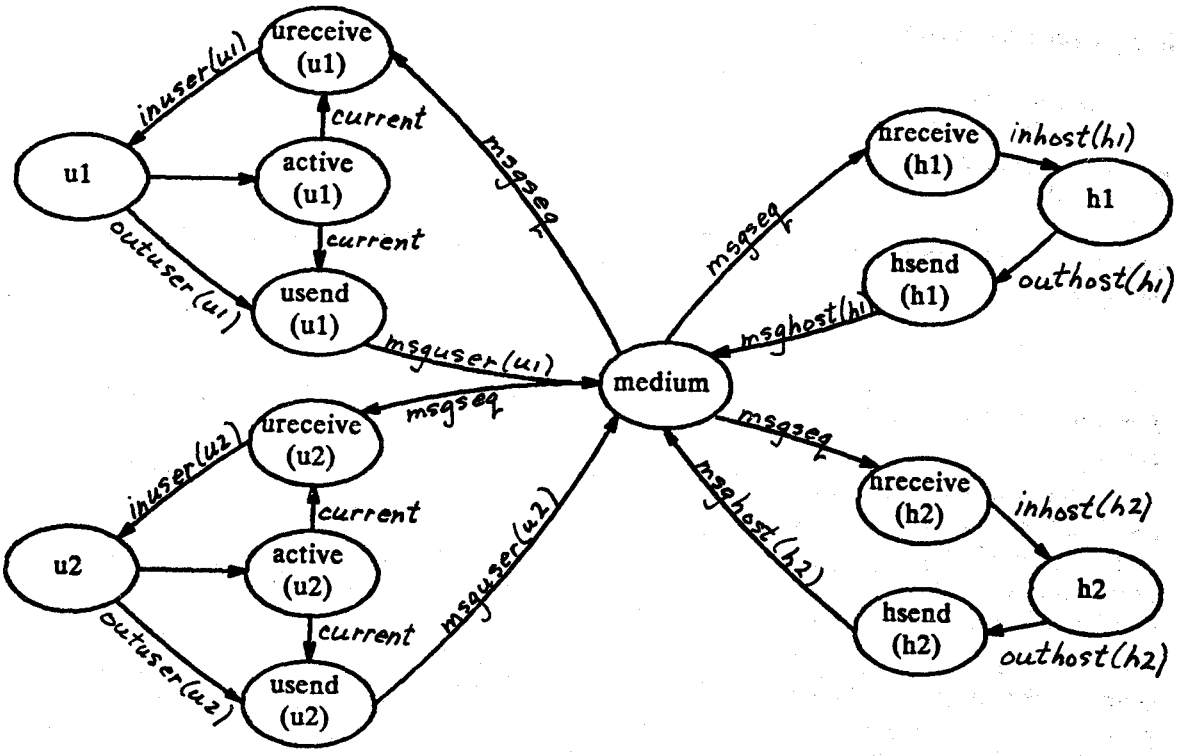


Figure 5. Example Dataflow Network

An example of a Lucid implementation is pLucid [Faustini83]. The data objects of this language consist of integers, reals, booleans, word character strings and finite lists. pLucid operations are made up of extensions of the usual arithmetic and logic operations as well as operations performed on lists. The network specification contained in this paper is implemented in pLucid.

A program in Lucid is simply an expression. Structuring of programs is achieved in a method similar to the where clause of Landin's ISWIM [Landin62]. Such a clause is of the form

E where
D_1
D_2
.
.
.
D_n

given an expression E and function definitions $D_1,...,D_n$. The "where" expressions can be nested to any depth. An example of a Lucid program that determines the sequence of

integer squares <0,1,4,9,...> is

```
square where
    int = 1 fby (1 + int);
    square = 0 fby int * int;
end
```

where the modal operation fby (read "followed by") is defined for any infinite sequences p and q as

$$p \text{ fby } q = \langle p_0, q_0, q_1, \ldots \rangle.$$

For further information on the language Lucid we refer you to the pLucid reference manual.

The Lucid specification of the dataflow network consists of two classes of specifications: node and configuration. First, each node on the dataflow graph must be formally specified. This is done by giving functional Lucid definitions that place the proper constraints on the flow of messages. We also attempt here not to place any unnecessary constraints on the system and thus over-specify the security.

To illustrate the specification of a node consider hsend, the processing module that sends messages from a host to the medium. We define hsend as a function of four input parameters: host,level,data, and destination. Level, data and destination are combined in a triple denoted message. This parameter is nil if no message is present at that time. Below is the Lucid definition of function hsend:

```
hsend(host,message) =
    if notpresent
        then nil
        elseif trusted(host)
            then [% host,lev,dat,destination %]
            else [% host,max(host),data,destination %]
    fi
        where
          lev = hd(message);
          data = hd(tl(message));
          destination = hd(hd(tl(message)));
          notpresent = message eq nil;
        end;
```

Given any host and message, the function hsend produces the value nil (no value) if no message is input. Assuming a message exists and the host is trusted then hsend produces the original message. If the host is not trusted then the level of the message produced is the value max(host) where max is the constant function that determines the maximum level of a given host.

The second class of Lucid specifications is the definition of the network configuration. We break this down into definitions for the users and hosts, and a specification of the message medium for the particular network. The medium definition is the only one that varies from network to network since it describes the actual users and hosts of a particular system.

Since for all hosts (or all users) the processing modules are the same, we can give a general definition for the output of each node in the subnet for a user or a host. The

medium definition states the actual parameters that will be applied to these definitions. We initialize the value of all arcs to be ⊥ (undefined).

```
// Specification of user x (corresponds to Figure 3.)

    outuser(x) = ⊥ fby user(x);
    inuser(x) = ⊥ fby ureceive(x,msguser(x),current(x));
    msguser(x) = ⊥ fby usend(x,outuser(x),current(x));

// Specification of host y (corresponds to Figure 4.)

    outhost(y) = ⊥ fby host(y);
    inhost(y) = ⊥ fby hreceive(y) ;
    msghost(y) = ⊥ fby hsend(y,outhost(y));

// Specification of the medium for a two user (u1,u2)
// and two host (h1,h2) network (corresponds to Figure 5.)

    msgseq = ⊥ fby medium(msghost(h1),msghost(h1),msguser(u1),msguser(u2));
```

For the configuration of Figure 5. we have assumed that host(h1), host(h2), user(u1) and user(u2) are all external message inputs to the system.

The Lucid specification also contains several auxiliary definitions that specify the constant mappings and the dominates relation. For example

```
clearance(x) = case x of
    "u1": 2;
    "u2": 3;
    default : 0;
end;
```

4.3 Security Model

In this section we express the constraints of Section 2 in a representation that corresponds to that of the Lucid/dataflow specification. Before this can be done we first extend the SNet security model to incorporate the notion of the state of a network.

The set A of ***arcs*** for a given dataflow network contains all communication channels (arcs) on the corresponding graph. For example A = {inuser(u1),inuser(u2),msgseq,inhost(h1),...} for the network in Figure 5.

We consider the ***state*** of a network to be a list of the values of arc set A at some time i. That is, if A = {a1,a2,...,an} then s_i = [$a1_i$,$a2_i$,...,an_i]. These correspond to the ith values of the Lucid history sequences for the arcs as described in Section 4.2.

The introduction of the notion of state and of time in this way is an artifact of the specification. What we are doing is describing the processing nodes as if they operate in strict global synchronization which, of course, they do not. By doing this, however, we have a framework in which to describe constraints on certain events that are partially ordered. We are only interested in these partially ordered events and the constraints.

In the general network constraints model of Section 2 we have the concepts of send and receive events. We relate these events to a given state as follows:

The set of ***send*** events for a network in state s_i (denoted se_i) is

$$(\bigcup_{u \in U} \text{msguser}(u)_i) \cup (\bigcup_{h \in H} \text{msghost}(h)_i)$$

The set of ***receive*** events for a network in state s_i (denoted re_i) is

$$(\bigcup_{u \in U} \text{inuser}(u)_i) \cup (\bigcup_{h \in H} \text{inhost}(h)_i)$$

In both cases we consider these events as a set of non-nil messages, i.e. msgset $\cup$ nil = msgset. Thus a send event is always of the form (sender,level,data,destination) and a receive event is of the form (receiver,level,data).

We say that a receive event (rec,lev1,dat1) in state s_i is ***related*** to a send event (sen,lev2,dat2,dest) in state s_j iff

rec = dest
lev1 = lev2
dat1 = dat2 and
$j < i$

The notion of relatedness does not necessarily imply cause, only possible cause. That is, a receive may have resulted from a related send.

Finally we define the ***currentlevel*** of a user u at time i, currentlevel$(u)_i$, as the level of the message being sent by user u at time i, i.e. if active$(u)_i$ = (host,level) then currentlevel$(u)_i$ = level. Also, the reader will notice that a constant in Lucid is treated as an infinite sequence of the same constant value.

Network Constraints

1. Users' levels
   For all $u \in U$ and all $s_i \in S$,
   currentlevel$(u)_i \leq$ clearance$(u)_i$

2. Labelling of messages
   a) For all $s_i \in S$ and all $h \in H$,
      If $(h,lev,dat,dst) \in se_i$ and not trusted$(h)_{i-1}$
      then lev = max$(h)_{i-1}$
   b) For all $s_i \in S$ and all $u \in U$,
      If $(u,lev,dat,dst) \in se_i$
      then lev = currentlevel$(u)_{i-1}$

3. Authenticity of Messages

$$(\exists s_{i_1},\ldots,s_{i_n} \in S)[(r \in re_{i_1} \wedge \ldots \wedge r \in re_{i_n}) \rightarrow$$
$$(\exists s_{j_1},\ldots,s_{j_n} \in S)(s_1 \in se_{j_1} \wedge \ldots \wedge s_n \in se_{j_n} \wedge$$
$$r \text{ in state } s_{i_k} \text{ is related to } s_k \text{ in state } s_{j_k}$$
$$\text{for all } k,\ 1 \leq k \leq n)]$$

4. Flow Control

a) For all $s_i \in S$ and all $h \in H$

If $(h,lev,dat) \in re_i$ then $lev \leq max(h)_i$

b) For all $s_i \in S$ and all $u \in U$

If $(u,lev,dat) \in re_i$ then $lev \leq currentlevel(u)_i$

4.4 Verification and Implementation

The language Lucid has powerful inference/manipulation rules [Ashcroft79] that allow for straightforward verification of Lucid programs. We can use these rules to prove that the Lucid specifications for any given network adhere to the security constraints. This is illustrated by the following proof for labelling of messages as stated in constraint 2a.

Constraint:

For all $s_i \in S$ and all $h \in H$

If $(h,lev,dat,dst) \in se_i$ and not $trusted(h)_{i-1}$

then $lev = max(h)_{i-1}$

Proof:

Assume for some state $s_i \in S$ and some host $h \in H$

$(h,lev,dat,dst) \in se_i$ and not $trusted(h)_{i-1}$

then $(h,lev,dat,dst) = msghost(h)_i =$

$\perp$ fby $hsend(host,message)_i =$

$hsend(host,message)_{i-1} =$

if notpresent

then nil

elseif $trusted(host)_{i-1}$

then [% host,lev,data,destination %]

else [% $host,max(host)_{i-1}$,data,destination %]

by definition of hsend. Since $(h,lev,dat,dst) \in se_i$ it must be the case that not (message eq nil), thus notpresent is false and therefore:

$hsend(host,message)_i =$

if $trusted(host)_{i-1}$

then [% host,lev,data,destination %]

else [% $host,max(host)_{i-1}$,data,destination %]

We have assumed not trusted(host) thus we get the result

$(h,lev,dat,dst) =$ [% $host,max(host)_{i-1}$,data,destination %]

therefore $lev = max(host)_{i-1}$.

5. Event-Based Specification

The event-based method we are using is based on the work of Chen and Yeh [Chen82,Yeh83a,Yeh83b]. Their specification language, called EBS, distinguishes between the ***behavior*** and the ***structure*** of a distributed system. The behavior is described in terms of externally visible effects of operations on a "black box" system comprising a single processing node; the structure describes the decomposition of this node into inter-connected

component processing nodes, each of which in turn has a behavior specification.

The external interface of a processing node is represented by a set of ***ports*** each of which comprises a sequence of events. An ***event*** is an instantaneous local state transition, such as the receipt or transmission of a message, which has an associated set of values determined at the time of its occurrence. The language is based on two fundamental relations on events: the ***precedes*** relation represents a partial time ordering, and the ***enables*** relation represents causality. Consequently, both control-related and data-related properties can be specified. It is important that no global state assumptions are required.

The top-level conceptual model of a distributed system from a user's point of view is a black box with only the interface visible. The user can only observe events in the top-level behavioral specification. A designer, however, views the system as a set of processes communicating via communication channels as described in the structural specification.

There are two ways of connecting component processes, or subsystems, in a structural specification: an ***interface*** connects a port of a subsystem with a port of the enclosing system; a ***link*** connects a port of a subsystem with a port of another subsystem. For both kinds of connection any event occurring in the first port occurs simultaneously in the connected port.

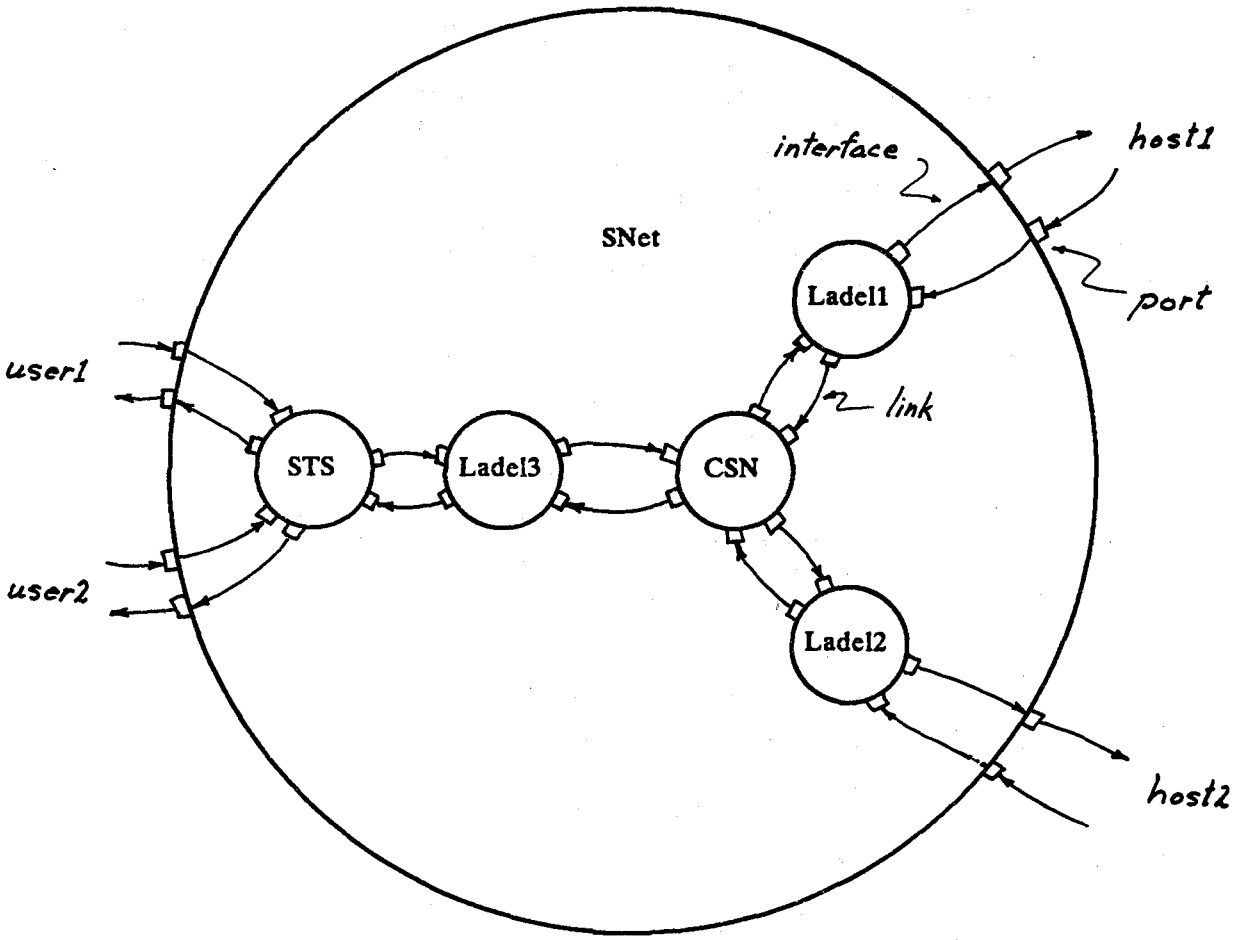


Figure 6. EBS Structural Specification of SNet

Each internal process is further refined in lower levels of specifications, as a set of subprocesses which again communicate among themselves with well defined communication channels. These subprocesses can be further decomposed until the desired level of detail is reached. This decomposition process allows for the use of the specification method with hierarchical design methodologies.

Figure 6. shows a graphical representation of a top-level structural specification for SNet.

5.1 The VEBS Specification Method

EBS, unfortunately, is not supported by a language processor and verification system. Since an automated system greatly assists the specification and verification of any practically sized system, we opted for a compromise; our method, VEBS, uses the commercially available VERUS specification language [Marick83] to express the semantics of EBS. As a preliminary step to the work on SNet, the expressibility of EBS concepts with VERUS was tested using an example of the Alternate-Bit-Protocol given in [Chen82]. Some of the proofs were carried out using the VERUS theorem prover-checker.

VERUS is a package of three programs: a parser to check specifications syntactically, an unparser for formatted output of parsed specifications, and a theorem prover-checker to check consistency of specifications. The VERUS specification language is not tied to any one application, e.g., state machine specifications, but is general enough for a wide variety of applications and provides a notation for first-order predicate calculus with types and functions. The only primitive types are integers and booleans; all other types must be defined by the user. The definition of structured types is somewhat awkward. For instance, the definition of a message type as an event in EBS is done in a notation similar to Pascal:

```
outport.msg:
  record
    sender: hostid;
    contents: data;
  end;
```

The notation outport.msg illustrates the fact that a port can accommodate only one event type. In VERUS notation this is expressed:

```
TYPE  EVENT;
TYPE  HOSTID;
TYPE  DATA;
VAR   sv              : HOSTID;
VAR   cv              : DATA;
DECLARE    Sender(EVENT) : HOSTID;
DECLARE    Contents(EVENT) : DATA;
DEFINE     Event_Record(sv, cv) : EVENT
BY
   AND
   {
     (Sender(Event_Record(sv, cv)) = sv);
     (Contents(Event_Record(sv, cv)) = cv);
   };
```

A reference to a field of a record in EBS notation, e.g. the sender field of an event variable ev of type outport.msg, is illustrated by:

ev.sender,

while in VERUS it will be:

Sender(ev)

As illustrated above, terms in VERUS are always typed. New types are defined in terms of the primitive data types, as well as of user defined types. One can declare enumerated data types; the elements are ordered by their occurrence in the declaration and can be used with relational operators. In general, a term can be either a constant, a variable, or a function reference.

The basic entities in VEBS are infinite sequences of events. In our case, events are associated with messages sent or received. The events in a behavioral specification belong to one of three sets: PORTS, a set of unidirectional communication channels between the process and its environment; SYS, a set of events internal to the process; and ENV, a set of events in the environment of the process.

Information about events is retrieved using access functions; for example, the time-stamp of an event in a port is returned by the function TimeStamp(port, event). Events in a port are totally ordered. All the semantics of EBS constructs are provided to the VEBS specification in the form of axioms (predicates known to be true).

The following naming convention is observed. Functions and constants begin with an upper case letter. Variables begin with a lower case letter. Types are entirely upper case.

5.2 VEBS Specification of SNet

In this section we give some examples of part of a second level specification for a two-user, two-host SNet system. The specification occurs in two parts, the structural specification and the behavioral specification for the component processes. The top-level behavioral specification, of course, represents the security model. The complete specification will appear in [Mercouris84].

The subjects of the model, users and hosts, are constants in the VEBS specification; User1, User2, Host1, and Host2. Each port is associated with a single subject only; therefore we can conceive the constraints applying to users and hosts, as constraints on the events occurring in the associated ports. The events are the objects being manipulated by the subjects. Users of SNet communicate with the network through unidirectional ports named like the following: SNet_user1_in, and SNet_user1_out for communication between User1 and SNet. All the ports are named in the same fashion. They consist of three parts separated by an underscore; the first part of the name denotes the subsystem where the port belongs, the second the communicating partner, and the third denotes an ***inport*** or an ***outport***. Inports are used to direct messages from the environment into the processing node, and outports from the processing node to the environment. Likewise, processing nodes consist of two parts separated by an underscore: the first part denotes the subsystem specified, and the second is the letter "p". For instance, the processing node of the SNet system is denoted by SNet_p.

As described before, events can carry information. For instance, the function Contents_data returns the data portion of an event. Other functions give the security level of the data portion (Contents_level), the sender field (Contents_sender), and the receiver field (Contents_receiver).

The variables of type USER, or HOST in the functions in the informal model have been replaced by the associated port variables in the formal expression of both the security model and the specification. Currentlevel additionally contains the time as a parameter in

its argument list. The functions used in this specification, as well as in the formal expression of the security model, have the same name as the corresponding ones in the informal model.

Structural Specification

The structural specification consists of declarations of the subsystems of which a system is composed, and how these subsystems communicate. This particular SNet is composed of one Sts (Secure Terminal Server), three Ladel devices, and the communications subnetwork Csn. The decomposition of the system is expressed with:

```
DEFINE System_Structure_SNet : BOOLEAN
BY
  AND
  {
    InSystem(SNet, Sts);
    InSystem(SNet, Ladel1);
    InSystem(SNet, Ladel2);
    InSystem(SNet, Ladel3);
    InSystem(SNet, Csn);
  };
```

where InSystem(SNet, Sts) states that any event that occurs in Sts occurs in the SNet system as well.

Likewise, we can specify ports and process nodes to be a part of SNet. In particular, the SNet system as seen in the behavioral specification consists of a processing node, SNet_p, and a number of ports. The statement of this structure is:

```
DEFINE System_Components_SNet : BOOLEAN
BY
  AND
  {
    InSystem(SNet, SNet_p);
    InSystem(SNet, SNet_user1_in);
    InSystem(SNet, SNet_user2_in);
    InSystem(SNet, SNet_user1_out);
    InSystem(SNet, SNet_user2_out);
    InSystem(SNet, SNet_host1_in);
    InSystem(SNet, SNet_host2_in);
    InSystem(SNet, SNet_host1_out);
    InSystem(SNet, SNet_host2_out);
  };
```

The definition of links for SNet is:

```
DEFINE Links_SNet : BOOLEAN
BY
  AND
  {
    Link(Sts_ladel_in, Ladel3_sts_out);
    Link(Sts_ladel_out, Ladel3_sts_in);
    Link(Ladel3_csn_in, Csn_ladel3_out);
    Link(Ladel3_csn_out, Csn_ladel3_in);
    Link(Ladel1_csn_in, Csn_ladel1_out);
    Link(Ladel1_csn_out, Csn_ladel1_in);
    Link(Ladel2_csn_in, Csn_ladel2_out);
```

```
    Link(Ladel2_csn_out, Csn_ladel2_in);
};
```

Behavioral Specification

The ladel is responsible for enforcing the information flow policy by setting the level on transmission of a message, and by allowing reception of only those messages that conform with the information flow policy. These are known as the labelling and delabelling functions. The complete specification of the ladels requires additional properties, such as the fact that they do not generate messages internally. An example showing only the labelling/delabelling is shown below.

```
DEFINE Ladel1_delabelling : BOOLEAN
BY
  FOR event2 FOR event1
    IF AND
    {
        InObject(Ladel1_host1_out, event2);
        InObject(Ladel1_csn_in, event1);
        Enables(event1, event2);
    }
    THEN Contents_level(event1) <= Level(Ladel1);

DEFINE Ladel1_labelling : BOOLEAN
BY
  FOR event2 FOR event1
    IF AND
    {
        InObject(Ladel1_host1_in, event1);
        InObject(Ladel1_csn_out, event2);
        Enables(event1, event2);
    }
    THEN
        IF Trusted(Ladel1)
        THEN Contents_level(event2) = Contents_level(event1)
        ELSE Contents_level(event2) = Level(Ladel1);
```

where InObject(Ladel1_host1_in, event1) tests if event1 has occurred in the history of the Ladel1_host1_in port. The expression Enables(event1, event2) means that the occurrence of event1 caused the occurrence of event2. It must also be specified that Level(Ladel1) = Max(Host1), and Trusted(Ladel1) = Trusted(Host1).

5.3 Security Model

We first express the constraints in a set of predicate definitions to simplify the specification. These definitions are used in the formal specifications and in the expression of the model in VEBS. For instance, the definition of the first constraint is:

```
DEFINE Constraint1(net_user_in): BOOLEAN
BY
  FOR t
    Currentlevel(net_user_in, t) <= Clearance(net_user_in);
```

net_user_in is a variable of type INPORT, and t of type TIME. Since each port is

associated with a single user only, then the value of Clearance(net_user_in) must be specified elsewhere to be equal to Clearance(u), where u is the user associated with the net_user_in inport. Similar statements apply to Currentlevel, Trusted, and Max. Of course, net_user_in is only a formal parameter here; the actual parameter is supplied when the constraint is applied to a particular configuration.

Recall that the informal model associated the execution of operations with events. Here, we view the execution of an operation as an event in a network inport and associate that event with an internal event in the network. This association is made explicit in the second constraint on labelling messages. In VEBS this is expressed:

```
DEFINE Constraint2a(net_user_in, net_p) : BOOLEAN
BY
  FOR event1
    IF InObject(net_user_in, event1)
    THEN EXIST event2
        AND
        {
          Enables(event1, event2);
          InObject(net_p, event2);
          Contents_level(event2) =
            Currentlevel(net_user_in,TimeStamp(net_user_in, event1));
        };


DEFINE Constraint2b(net_host_in, net_p) : BOOLEAN
BY
  FOR event1
    IF InObject(net_host_in, event1)
    THEN EXIST event2
        AND
        {
          Enables(event1, event2);
          InObject(net_p, event2);
          IF Trusted(net_host_in)
            THEN Contents_level(event2) = Contents_level(event1)
            ELSE Contents_level(event2) = Max(net_host_in);
        };
```

where net_p is a variable of type PROCESS, and net_user_in is a variable of type INPORT.

The third constraint is expressed in two parts: part a) defines the authentication and integrity property, and part b) addresses the non- duplication of messages. Constraint four follows.

```
DEFINE Constraint3a(net_name_in, net_name_out) : BOOLEAN
BY
  FOR event2 FOR l FOR d
    IF AND
    {
        InObject(net_name_out, event2);
        Contents_receiver(event2) = net_name_out;
        Contents_level(event2) = l;
        Contents_data(event2) = d;
    }
```

```
    THEN EXIST event1
        AND
        {
          InObject(net_name_in, event1);
          Contents_receiver(event1) = net_name_out;
          Contents_level(event1) = l;
          Contents_data(event1) = d;
          Enables(event1, event2);
        };

DEFINE Constraint3b(net_name_in, net_name_out): BOOLEAN
BY
  FOR event1
    IF InObject(net_name_in, event1)
    THEN IF EXIST event2 EXIST net_name_out
          AND
          {
            InObject(net_name_out, event2);
            Enables(event1, event2);
          }THEN FOR event2
            NOT EXIST event3 NOT EXIST net_name1_out
                AND
                {
                  InObject(net_name1_out, event3);
                  Enables(event1,event3);
                  NOT event2 = event3;
                  NOT net_name_out = net_name1_out;
                };

DEFINE Constraint4a(net_host_out) : BOOLEAN
BY
  FOR event1
    IF InObject(net_host_out, event1)
    THEN Contents_level(event1) <= Max(net_host_out);

DEFINE Constraint4b(net_user_out) : BOOLEAN
BY
  FOR event1
    IF InObject(net_user_out, event1)
    THEN Contents_level(event1)
              <= Currentlevel(net_user_out, TimeStamp(net_user_out, event1));
```

where l is a variable of type LEVEL, and d a variable of type DATA. Also, $net\_name_i\_in$ is a variable that may assume the value of any inport of the SNet system, user or host. Likewise the $net\_name_i\_out$ can assume the value of any outport of the SNet.

The subjects of the model, except SNet_p, are variables in VEBS. SNet_p is constant because it refers to a single processing node. An arbitrary number of users and hosts may communicate using SNet_p, and that is why the associated ports are represented by variables. Users of the SNet system connect to ports named $snet\_user_i\_in$, and $snet\_user_i\_out$. Likewise, host ports are named $snet\_host_i\_in$, and $snet\_host_i\_out$. However, snet_user_in is a variable of type INPORT, and may assume the value of any inport associated with a user. This is also true for snet_user_out, snet_host_in, and snet_host_out. The operations in the informal security model are expressed by events occurring at the

user and host interface ports. The events of the informal security model are internal events occurring in SNet_p.

We express the constraints of the security model for an arbitrary number of users and hosts using the definitions shown above. We express the constraints in terms of relationships on events happening in SNet inports and outports.

```
DEFINE Mod_Constraint1 : BOOLEAN
BY
  FOR snet_user_in
    Constraint1(snet_user_in);

DEFINE Mod_Constraint2a : BOOLEAN
BY
  FOR snet_user_in
    Constraint2a(snet_user_in, SNet_p);

DEFINE Mod_Constraint2b : BOOLEAN
BY
  FOR snet_host_in
    Constraint2b(snet_host_in, SNet_p);

DEFINE Mod_Constraint3a : BOOLEAN
BY
  FOR snet_name_in FOR snet_name_out
    Constraint3a(snet_name_in, snet_name_out);

DEFINE Mod_Constraint3b : BOOLEAN
BY
  FOR snet_name_in FOR snet_name_out
    Constraint3b(snet_name_in, snet_name_out);

DEFINE Mod_Constraint4a : BOOLEAN
BY
  FOR snet_host_out
    Constraint4a(snet_host_out);

DEFINE Mod_Constraint4b : BOOLEAN
BY
  FOR snet_user_out
    Constraint4b(snet_user_out);
```

Now using the above expression of the model the top-level specification becomes concise and particularized to the configuration specified. For example,

```
DEFINE Constraint2a_sp : BOOLEAN
BY
  AND
  {
    Constraint2a(SNet_user1_in, SNet_p);
    Constraint2a(SNet_user2_in, SNet_p);
  };
```

5.4. Verification and Implementation

Verification of the specification using VERUS is now underway and will be the subject of a future paper.

There is some work being done to complement the abstract EBS specification language with CSP as an implementation language [Yeh83b]. There are two reasons why CSP is not suitable for us. First, communication between two parties takes place only when they are both ready, and that constitutes a bidirectional information flow contradicting the unidirectional flow requirement. Second, our prototype work is in Concurrent Euclid.

Verification between levels in the abstract specification will be done using the VERUS theorem prover-checker. Each process's behavioral specification must be verified against the decomposition of that process in the next lower level to a set of behavioral requirements for each subprocess and a structure for these subprocesses.

Verification of the implementation has not been addressed. We hope to transform the specification into a set of predicates that can be used in an informal comparison with the CE code.

6. Summary

Developing the security model along with the specifications has been extremely valuable in sharpening our understanding of the security model. In an ideal world one defines the security model in isolation so that it is unencumbered with dependencies on particular systems or specification methods. In practice, however, we found the security model very difficult to get right. Some revisions to it resulted from simply reviewing it, but others resulted directly from attempts to specify the constraints that directed us to focus on problems.

This experience in trying to get the model right and going through many revisions, of course, leaves us with some doubts as to its appropriateness even now. The problem, of course, is that our model is very operational in its nature. We have spent some time in trying to find a more abstract model that will capture the constraints that we want. Thus far we have not been successful but this effort will continue.

References

[Ashcroft79]
Ashcroft, E.A., Wadge, W.W.,
Structured Lucid,
University of Waterloo Technical Report CS-79-21,
June, 1979.

[Ashcroft83]
Ashcroft, E.A., Wadge, W.W.,
Why Lucid,
Distributed Computing Project No. 3,
Department of Computer Science,
University of Warwick,England,1983

[Chen82]
Chen, Bo-Shoe,
Event-Based Specification and Verification of
Distributed Systems,
Department of Computer Science,
Ph.D. Thesis, University of Maryland, 1982.

[Faustini83]
Faustini, A.A., Mathews, S.G. and Yaghi, A.A.,
The pLucid Programming Manual,
Department of Computer Science, University of Arizona,
Tempe, Arizona, 1983.

[Glasgow84]
Glasgow, J.I., MacEwen, G.H.,
Specification of a Distributed Multi-Level Secure System:
The Lucid/Dataflow Approach
Biennial Symposium on Communications,
Queen's University, Kingston, 1984

[Lamport78]
Lamport, L.,
Time, Clocks, and the Ordering of Events in a Distributed System
CACM 21, 7(Jul 78), 558-565.

[Landin62]
Landin, P.J.,
The Next 700 Programming Languages,
Proceedings of IFIP, 1962.

[MacEwen84a]
MacEwen, G.H., Burwell, Bruce, Lu, Zhuo-Jun,
The Design of a Distributed Multi-Level Secure System
Based on Physical Isolation,
IEEE Symp. on Security and Privacy, Oakland,CA, April 84.

[Marick83]
Marick, B., Mostek, J.W., Wagner, F.R.,
VERUS Language Manual,
Compion Corporation, January, 1983.

[Mercouris84]
Mercouris, T.
Specification of a Distributed Multi-Level Secure System
M.Sc. Thesis, in preparation,
Queen's University, Kingston

[Yeh83a]
Yeh, R.T., Chen, Bo-Shoe,
Formal Specification and Verification of
Distributed Systems,
IEEE Trans. on Soft. Eng. SE-9, 6(Nov 83), 710-722

[Yeh83b]
Yeh, R.T., Reed, J.N.,
Specification and Verification of Liveness
Properties of Cyclic Concurrent Processes,
Computer Science Department,
University of Maryland, March, 1983.

A MULTI-LEVEL SECURE LOCAL AREA NETWORK

Albert L. Donaldson

Verdix Corporation
7655 Old Springhouse Road
McLean, Va. 22102

This paper describes a local area network (LAN) architecture that utilizes *Network Security Devices* (NSDs) to provide trustworthy interfaces between untrusted users and a multi-level secure network. The design is based on concepts defined in the *DoD Trusted Computer System Evaluation Criteria* (TCSEC) [DoDCSC] for Class A1 systems, and provides mandatory and discretionary access control mechanisms within the framework of a communications protocol reference model. The paper does not attempt to define trusted network evaluation criteria, but instead focuses on the requirements analysis and engineering integration of a *Trusted Network Base.*

1. Introduction.

Until recently, the formal security methods used to design secure computer systems have not been sufficiently integrated with LAN architectures to develop trusted LAN components. Formal specification and verification has been used in the development of highly-critical one-of-a-kind communications guard devices, such as the Message Flow Modulator and the Restricted Access Processor. However, these are not general purpose communications systems and do not use Department of Defense (DoD) or industry standard communications protocols.

The initial emphasis within the DoD has been on the need for trusted computer *systems*, rather than trusted computer *networks*. Last year the DoD Computer Security Center (DoDCSC) released the Trusted Computer System Evaluation Criteria (TCSEC), which identifies specific security features and assurance requirements to be used in the evaluation of commercial Automatic Data Processing (ADP) systems. There are two main reasons for this emphasis on trusted ADP systems. First is the high cost of physical, procedural and personnel security measures required to protect information in System High mode. Second, emerging DoD applications require interoperability between entities at different security levels, in strict accordance with security policy. These requirements can best be satisfied by trusted computer systems operating in Multi-Level Security mode.

A distributed approach to multi-level security has been proposed in [Rushby], which relies on a relatively simple but strategically placed guard device (the Trusted Network Interface Unit) to enforce an access control policy between untrusted computer systems. While this approach uses LAN components for interconnecting the host systems, the effort was directed toward developing a distributed multi-level computer *system* rather than a trusted *LAN.*

Our approach differs from previous work in that its purpose is to develop *trusted LAN interface components* to meet the emerging communications requirements of the DoD, in programs such as the WWMCCS Information System (WIS) [Bernosky, Gomberg]. Current commercial LANs do not provide the necessary communications protocols, security or performance required for interoperability between heterogeneous host systems, possibly operating at different security levels. Usage of DoD and industry standard protocols is essential to meet the DoD's interoperability requirements, while the usage of formal design methods is necessary to provide a high degree of assurance that the LAN components operate according to DoD security policy.

The remainder of this paper describes the architectural approach and communications protocol hierarchy, discusses the relevance of TCSEC to our Secure LAN design, and discusses the security requirements.

2. Secure Network Architecture.

The solution to the DoD's requirements for secure communications networks has been viewed in the past as a *network of trusted nodes,* consisting of independent trusted ADP systems, interconnected by encrypted communications links. This seems to be a quite reasonable approach to network security, until one considers the following two problems. First, each of the individual nodes is operated by a different security administrator, which would make the management of large, highly interconnected networks very tedious if not impossible. Second, in order for any particular host to handle multi-level data, the entire communications protocol hierarchy must be trusted. This approach is quite expensive, given the complexity of communications protocol software and the necessity of verifying all the protocol layers handling multi-level data.

Both of these problems can be solved by building a distributed *Trusted Network Base* (TNB), responsible for reliably enforcing access to the network in accordance with a well-defined network security policy. This approach allows a coherent policy to be controlled by a single network security administrator. More important, however, is that this approach requires only the lower protocol layers (*i.e.*, the TNB) to be trusted. Since all communications must pass through these lower layers, it is not necessary to trust the higher protocol layers or the host operating system.

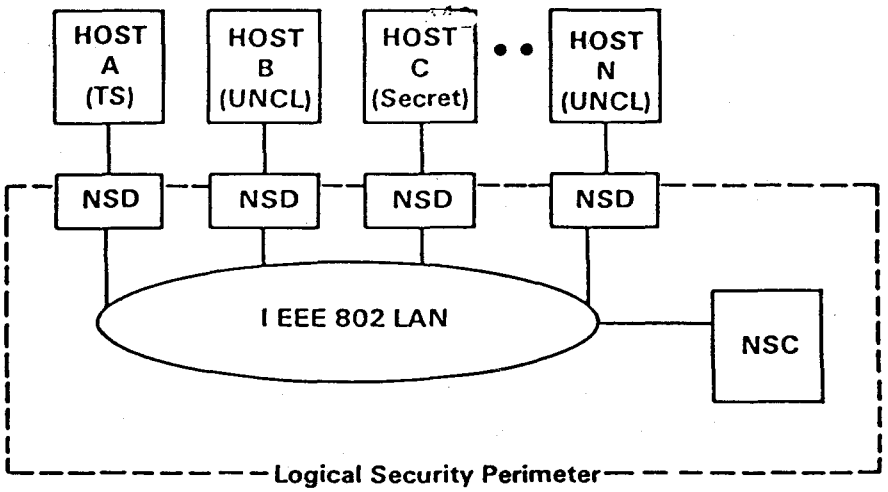


Figure 1. *Secure LAN Architecture.*

The top-level architecture shown in Figure 1 allows users with multiple security levels to share a single IEEE 802 LAN medium by using trustworthy *Network Security Devices* (NSDs) that strictly enforce a *Network Security Policy* and provide end-to-end encryption. A *Network Security Center* (NSC) provides capabilities for a network security administrator to control and audit the security aspects of the entire network. An untrusted *Host Specific Interface* (HSI) (not shown in the figure) can be used to convert the user's interface to the NSD's external interface, so that the NSD can be used without modification for different host and terminal devices.

The individual NSDs mediate the flow of user-labeled data based upon a security window (mandatory access controls) and an access matrix (discretionary access controls), both defined by the network security administrator at the NSC. The security window can be closed down to a single authorized classification and category, or can be opened up to accommodate multi-level secure hosts. Eight hierarchical classifications and 32 non-hierarchical categories are provided. As shown in Figure 2, packets (a, b, c) pass through the origin NSD's transmit window, while packets (d, e) are labeled below allowable limits and will be rejected. All data on the network is labeled with the user (or HSI) originated label, provided that it passes through the transmit window. At the destination, only packets (b, c) pass through the receive window, while packet (a) is labeled above allowable limits.

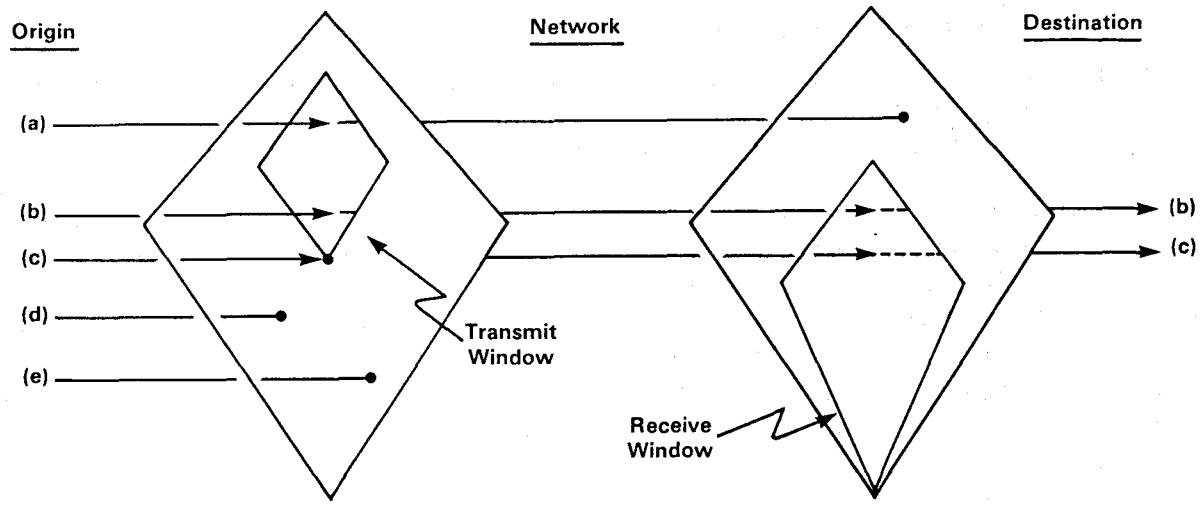


Figure 2. *Security Window Mechanism.*

The philosophy is that the network security administrator can decide whether a particular host's operation is sensitive enough to confine it to a single level and compartment, or if it can be trusted to properly identify its level and compartment set within certain bounds. This approach provides maximum flexibility and security while also accommodating trusted multi-level secure hosts. Properly labeled and properly addressed packets are routed transparently to the named destination, while improperly labeled or improperly addressed packets are intercepted and explicitly routed to the NSC for auditing.

3. Communications Architecture.

Network security must be discussed in the context of a communications protocol reference model, such as the International Standards Organization (ISO) Open Systems Interconnection (OSI) model [ISO] or the recent DoD Protocol Reference Model (PRM) [Ennis]. The PRM is based on [ISO], but was influenced by the ARPA Internet project and specifically addresses the DoD's future operational requirements. There are several differences between the two models, but the major distinctions center on the definition of *protocol layers* and the importance of internetworking. The OSI model defines seven protocol layers, each responsible for a specific abstract function, while the PRM is concerned primarily with the requirement that protocols be arranged hierarchically. The PRM identifies separate *network* and *internet layers*, which reside in a single *network* layer in the OSI model.

Since these issues primarily affect higher-level protocols outside the trusted network base, the following discussion is presented using the more familiar OSI reference model. Table 1 shows that the trusted access control mechanisms have been placed at layer 3 of the OSI reference model, between the IEEE 802 LAN protocols and the DoD Internet Protocol (IP). There are two reasons for this placement. First, local area network security measures properly reside in this layer, just as measures to enforce inter-network security properly reside within the IP. Second, this network layer is generally unused in IEEE 802 LANs, which allows the integration of security with the protocol structure rather than attempting to add security to existing protocols.

Table 1. Secure LAN Protocol Hierarchy.

OSI Layer	Protocol
4 - Transport	DoD Transmission Control Protocol (TCP)
3 - Internet	DoD Internet Protocol (IP)
---------- *LAN Logical Security Perimeter* ----------	
3 - Network	Network Security Protocol (NSP)
	Encryption
2 - Link	IEEE 802.2 Logical Link Control (LLC)
2 - Link	IEEE 802.3 Media Access Control (MAC)
1 - Physical	IEEE 802.3 Media Access Unit (MAU)
0 - Medium	baseband or broadband cable, fiber optics

Because we are proposing the development of a Secure LAN (and not a secure internetwork), we have drawn a *logical security perimeter* between the LAN security critical protocols and the higher-level internetworking protocols. Within this perimeter, we intend to specify and develop a Network Security Protocol (based on the TCSEC mandatory and discretionary access control requirements) that will allow only authorized LAN interactions. The Network Security Protocol and higher level data will be encrypted, leaving only the IEEE 802 protocol headers in the clear.

Usage of DoD TCP and IP is mandatory for "all DoD packet-oriented data networks which have a potential for host-to-host connectivity across network or subnetwork boundaries" [DeLauer]. However, these protocols are not part of the TNB. Their presence is required only for internetwork communications purposes, and not for security reasons.

The following paragraphs describe how the Secure LAN handles data labeling by untrusted users, and how data is transferred from one security level to a higher level.

IP security label. According to the *Internet Protocol Transition Workbook* [Postel],

> TCP modules which operate in a multilevel secure environment must mark outgoing segments with the [user's] security, compartment, and precedence. Such TCP modules must also provide to their users or higher level protocols such as Telnet or THP an interface to allow them to specify the desired security level, compartment, and precedence of connections.

This information is copied into the IP Security Option field, and evaluated by the destination host. Note that [Postel] does not define what is done if the attempted transfer violates the security policy, and also leaves unstated the accreditation requirement for a trusted host to process data at multiple security levels.

However, untrusted hosts can be used in a multi-level secure environment if access to the network is controlled by a TNB. The NSD provides an independent check on the TCP-specified labeling by correlating it against information defined by the network security administrator. Properly labeled IP-data-units are transferred into the network, while improperly labeled data is intercepted and logged at the NSC.

Low-to-high data transfers. High-level protocols such as the File Transfer Protocol (FTP) are full-duplex, with data flowing in one direction and acknowledgements in the other. For example, a file transfer from an Unclassified host to a Secret host requires acknowledgements to be transferred from the Secret host to the Unclassified host. Because this conflicts with multilevel security policy, [Rushby] concludes that the best way to provide secure information flow across security boundaries is through a *trusted* intermediary, the Secure File Store.

While the usage of such a trusted intermediary is compatible with our security window approach (*i.e.*, it is simply another host), we have chosen to view this as a host function requiring a trusted multi-level secure host. (This seems reasonable since the NSD operates at the network layer, while

file transfers are a higher-level host function.) Either the Secret host must be trusted to *downgrade* the acknowledgement to Unclassified, or the Unclassified host must be trusted to properly handle the Secret acknowledgement. Depending upon the approach, either the Secret host's transmit window or the Unclassified host's receive window must be opened sufficiently to allow the acknowledgement to pass through.

4. Trusted Network Evaluation Criteria.

There has been considerable debate over the applicability of the TCSEC to secure computer networks. The TCSEC explicitly addresses the development of *trusted general-purpose and trusted embedded ADP systems* and does not mention any applicability to network configurations. While the DoDCSC is currently involved in the development of network criteria, informal discussions lead us to believe that it may be a year or more before they are approved and released. This leaves secure network developers in a difficult position, since two reasons for such criteria are:

> To provide users with a metric with which to evaluate the degree of trust that can be placed in computer systems for the secure processing of classified and other sensitive information.
>
> To provide guidance to manufacturers as to what security features to build into their new and planned commercial products in order to provide widely available systems that satisfy trust requirements for sensitive applications. [DoDCSC]

Contemporary thinking is that there are fundamental differences in nature (*e.g.*, functions, relationship between subjects and objects, protection issues) between computer systems and communications systems that necessitate the development of a separate set of *trusted network evaluation criteria.* The primary difference is that users can execute arbitrary programs on computer systems, while communications systems are generally used only for executing pre-defined protocol processes. In other words, computer system subjects (acting on behalf of system users) can interact in an almost unlimited number of ways, while communications systems consist of a fixed number of protocol process subjects whose actions and interactions are pre-defined by the hierarchical nature of the protocol reference model. Further, processes in a communications system tend to exist only as subjects, while they may exist either as subjects or objects in a computer system. A less obvious difference identified in the *Draft Communications Network Criteria* proposed by Ford Aerospace is:

> Processes (subjects) are relatively transitory and objects are relatively static in ADP systems, while the opposite is true for communications systems (*e.g.*, objects are transitory and processes are static). [FACC]

For these reasons, we do not believe the TCSEC is suitable in its existing form for the *formal evaluation* of secure LAN products, but we believe that the existing TCSEC security policy, accountability, and assurance requirements can be used *to provide guidance* toward the development of a Secure LAN.

The distinction between trusted distributed ADP *systems* and trusted *networks* is not always a clear one. For example, it seems quite feasible to build a LAN-based distributed secure computer *system* (such as that proposed by Rushby) according to the TCSEC, and have it evaluated by the DoDCSC at the A1 level. Because of these similarities, we believe the major differences between the existing TCSEC and the forthcoming trusted network criteria will be in the following areas:

- location of security mechanisms within the protocol hierarchy
- addressing network protection and encryption issues
- distinction between network policy and node policy.

The purpose of this paper is not to propose yet another set of criteria, but rather to explain how we intend to solve a security problem, based on our use of the TCSEC and its rationale.

5. LAN Security Requirements.

The LAN security requirements were derived from a threat analysis for IEEE 802 LANs, and from the TCSEC security policy, accountability, and assurance requirements for Class A1 systems.

5.1. LAN Threat Analysis.

Threat analyses are performed during the accreditation of operational sites, but are not normally done during trusted system development. However, because certain network threats are not addressed by the TCSEC, a LAN threat analysis was performed. It concluded that an integrated approach, utilizing both trusted logical mechanisms and intra-network encryption, is necessary to protect against different classes of security threats.

Network security threats can be categorized as either *external* (*e.g.*, wiretap), or *internal* (improper behavior by valid users and software). Different LANs have slightly different vulnerabilities with respect to external threats, but all LANs have the same vulnerability to internal threats.

External threats. Much of the existing work categorizing external security attacks is applicable to LANs as well as to wide area networks. Wiretappers can attack LANs using either passive or active techniques, depending on whether the goal is to eavesdrop or to modify data. The three different IEEE 802 LANs (CSMA/CD, token bus, and token ring) are equally vulnerable with respect to passive eavesdropping, since each of the access methods provides for complete coverage of the medium. Ring topologies are slightly more vulnerable than bus topologies with respect to active modification threats. With bus topologies the data is transmitted only once, and it cannot be easily modified enroute to its destination. Because of the multiple linkwise transmissions on a token ring, there is additional risk that data might be modified by an intermediate node or by a wiretapper between two legitimate nodes. External threats are best controlled by physical separation of the sensitive data from the outside world; however, encryption may be used when such physical measures are inapplicable. Another solution is fiber optics, which does not emanate RF energy, and is relatively difficult to tap without major disruption.

Internal threats. For trusted systems we must broaden our analysis to include those legitimate users who either maliciously or unintentionally release or modify information intended for other users. All three IEEE 802 LANs are equally susceptible to these internal threats, since all are designed to promote open sharing of information. These threats may take the form of passive attacks (listening to all data on the LAN), or active attacks (*e.g.*, spoofing). Internal threats can only be countered with trusted access control software that reliably enforces a security policy.

5.2. TNB requirements

Although there was some interpretation involved in translating the TCSEC requirements for a *TCB* to a communications environment, Part II of the TCSEC was used as a guide in identifying the basic intent of the requirements. Because the essence of security is access control, the goal is to ensure that the user's ability to write information to the LAN and read information from the LAN is in accordance with security policy.

In this context, the term *user* applies to the protocol layer immediately outside the TNB and does not necessarily apply to the individuals using the attached system. (If the attached system is a single-user workstation, one may make some assumptions about the behavior of its user, but in general the attached system will be a multi-user untrusted host computer.)

5.2.1. Security Policy.

The *security policy* requirements for mandatory security controls, discretionary security controls, and marking are listed below:

(1) Each NSD shall determine that the user-supplied sensitivity level is consistent with the transmit security window (defined for that NSD) before transmission to the network. All user data that is not properly labeled shall be rejected.

(2) Each NSD shall examine all data arriving from the LAN, and shall transfer the data to the user only if the sensitivity label of the data is consistent with the receive security window defined for that NSD. All data not satisfying this check shall be rejected.

(3) Each NSD shall maintain a list of other NSDs to which it is allowed to send data and a separate list of NSDs from which it is allowed to receive data. All data transmissions (either from the user or the network) which attempt to violate these controls shall be rejected.

(4) Each NSD shall label each packet of data to be transmitted on the LAN with the valid user-specified sensitivity level and destination address of the data.

The transfer of a user protocol-data-unit from user **A** to user **B** takes place in two stages: user **A** to LAN, and LAN to user **B**. (Again, the two users are defined as the host-resident protocol layers immediately outside the logical security perimeter, *i.e.*, IP/TCP.) This may be modelled as a synchronized write-read transaction between the two user subjects, with subject **A** writing the object **O** to the LAN, and subject **B** reading the object **O** from the LAN.

Mandatory Access Control. The TCSEC control objective description for mandatory security policy states:

> Mandatory security refers to the enforcement of a set of access control rules that constrains a subject's access to information on the basis of a comparison of that individual's clearance/authorization to the information, the classification/sensitivity designation of the information, and the form of access being mediated. Mandatory policies either require or can be satisfied by systems that can enforce a partial ordering of designations, namely, the designations must form what is mathematically known as a *lattice*.

As defined in requirements 1 and 2, this is satisfied by the necessity for data to pass through the transmit security window (represented by a lattice structure) at NSD **A**, and through the receive security window at NSD **B**, before it can be delivered to the destination user. The transmit window prevents the origin user from labeling information below allowable limits, while the receive window prevents the destination user from reading information above its clearance level. These are similar to the Bell and LaPadula *-Property and Simple Security Condition, respectively.

Discretionary Access Control. The TCSEC provides the following description of discretionary security:

> The basis of this kind of security is that an individual user, or program operating on his behalf, is allowed to specify exactly the types of access other users may have to information under his control. Discretionary security differs from mandatory security in that it implements an access control policy on the basis of an individual's need-to-know as opposed to mandatory controls which are driven by the classification or sensitivity designation of the information.

There are two critical concepts within this description: first, it is *user specified* (hence the name *discretionary*), and second, it allows access on a *need-to-know* basis. The first issue is to some extent already satisfied by network mechanisms, since it is at the user's *discretion* whether to send or receive a data unit from the TNB. It is impossible for user **B** to read an object that is not written by user **A**; even if user **A** writes the object, user **B** is not forced to read it. Unless both users cooperate, the transfer will not be completed. We feel that the second issue is significantly more important, *i.e.*, the need for a finer grained access mechanism that operates on a need-to-know basis. Such a need-to-know concept can be reasonably extended to host systems, even though protocol addresses do not refer to specific individuals.

As described in Requirement 3, the proposed mechanism is via transmit and receive access control lists enforced by the NSDs. Based upon the observation that an object exists only in the context of a synchronized write-read transaction pair, the transmit list defines the allowable destination link addresses for objects that can be written to the network. Similarly, the receive list defines the allowable set of origin link addresses for objects that can be read from the network. Since these lists are defined by the network security administrator rather than by the individual users, the classical meaning of *discretionary* has been modified, but the intent of need-to-know access is preserved.

Marking. As noted in Requirement 4, proper marking of communications objects is necessary in order for the destination NSD to properly perform mandatory and discretionary access controls. This labeling requirement also includes the internal storage of the data units while they are being processed within the NSDs. The requirements for labeling human-readable output and interactive notification of a subject's sensitivity level do not apply within the context of a Secure LAN, because these are application level functions.

5.2.2. Accountability

Even though the TNB interacts directly with higher-level protocol entities rather than human users, it is still essential that there be an individual who is accountable for the proper operation of each NSD. For a single-user workstation, the accountable individual will probably be the individual using the workstation to access the network. For a large host computer, the accountable individual might be either the computer operator or host security administrator. However, this individual does not necessarily have to be a user of the system attached to the NSD.

User identification and authentication. User identification can be based on unique physical characteristics of the individual, what the user *has* in his possession, or what the user *knows*. The recommended approach is the usage of physical key or card devices in the possession of legitimate users, since such devices as fingerprint readers are expensive and password schemes are subject to attack. Because this is a *local* area network, these physical devices can be re-programmed and re-distributed as frequently as necessary.

(1) The NSD shall provide a trusted path between itself and the user identification hardware, and between itself and the NSC, in order to authenticate the individual who is responsible for the proper operation of the NSD.

Audit. Each NSD must notify the NSC of significant events such as startup attempts, rejected data, error conditions, and acknowledgements of NSC commands. The NSD must accumulate certain statistical information such as the number of data-units transmitted and received, and transfer this information periodically to the NSC. The NSC must log these events and must provide for notification of the network security administrator for selectable security-relevant events.

(1) Rejected data shall be logged, along with the identification of the transmitting or receiving NSDs, the reason for rejection, and a date and time stamp.

(2) Each NSD shall generate an audit trail of its actions and transmit it to the NSC.

5.2.3. Assurance

The operational and life-cycle assurance requirements are identified below.

Operational assurance. The key concept of operational assurance is that the Secure LAN TNB must maintain a protected execution domain that protects it from external influence by attached users or wiretappers. Protection is a two-layer requirement. First, the network facilities must be physically, procedurally and cryptographically protected in accordance with the maximum level of data on the network. Second, each of the individual NSD nodes must provide their own protected execution domain, which includes requirements for isolation of protocol processes and data objects. The internal structure of the TNB must be organized in accordance with the layering inherent in the protocol hierarchy. The following specific requirements have been defined:

(1) The Secure LAN shall be physically, procedurally and cryptographically protected in accordance with the maximum level of data on the network.

(2) The only access to the physical network shall be through the NSD. The NSDs shall not be bypassed.

(3) The software that implements the above mandatory and discretionary controls shall reside within the TNB and shall not be modified during execution.

(4) The NSDs and NSC shall perform periodic checks of their integrity and proper operation in the secure network. If an NSD determines that it is operating in a degraded mode, it shall stop transmitting data, notify the network security administrator, and shall take itself offline.

The TNB must provide for a *network security administrator* position at the NSC, in order to define the mandatory and discretionary access controls for each user and to audit the operation of the network. The NSC must be dedicated to the operation of the Secure LAN. However, there is no requirement for separate operator and administrator positions (as required for the higher TCSEC classes), since the operator functions are significantly reduced in a LAN environment. Invalid operation of any portion of the secure network (including rejected data) must cause an alarm to be generated.

Life-cycle assurance. The TNB must be designed and engineered using formal specification and verification tools, similar to the requirement for TCSEC Class A1 systems. This includes a security policy model, a formal top level specification, formal design verification, and implementation verification, in addition to security testing and configuration management.

6. Conclusions.

Trusted LAN communications products can be built that will integrate many of the security requirements stated in the TCSEC with existing DoD and industry standard protocols. We have found that the mandatory and discretionary access controls required by the TCSEC for trusted computer systems are directly applicable, and can be implemented within a network protocol, without modification to existing DoD and industry standard protocols.

The proposed architecture uses *Network Security Devices* (NSDs) to interface untrusted hosts to a multi-level LAN by enforcing mandatory and discretionary access controls on all transfers to and from the network. The approach has been to define a *Trusted Network Base* architecture that can implement a coherent security policy based on security windows and access control lists defined by the network security administrator at the *Network Security Center* (NSC). These trusted mechanisms provide an independent check on user supplied labeling, which represents a significant increase in security over existing hierarchical security labeling approaches, such as labeling of IP packets based upon information supplied by higher protocol layers.

Because the individual NSDs and the LAN medium may be located in unprotected environments, Verdix is providing intra-network end-to-end encryption using the DES for protection against external wiretapping. While the DES has not been approved for the protection of classified information, it is the only such VLSI device currently available. It is clear that there must eventually be a consensus with respect to approved, lower-cost encryption devices for LANs. Either the DES must be approved for handling some levels of classified data in moderately protected locations such as LANs, or a new, less expensive encryption device must be provided for that application.

Finally, while this architecture has immediate applicability as a trusted interface allowing untrusted hosts to be used in a multi-level secure environment, its usefulness does not end when trusted host systems become readily available. The NSD can still be used to offload protocol processing and provide an *additional degree of trust* (even with a network consisting of A1 hosts). The centralized control afforded by the NSC is essential to secure network management.

Acknowledgements

This work was partially funded by the Air Force Systems Command, Electronics Systems Division, under Contract F19628-83-C-0171, as part of the DoD Small Business Innovative Research (SBIR) program. The author also wishes to acknowledge the contributions of Karl Nyberg, George Cowan, Donn Milton, Steve Deller, Chetan Sanghvi and Jerry Shelton toward the design of this system.

References

[Bernosky] - Larry Bernosky and Bill Blankertz, "WIS: Gateway to Worldwide Military Communications", *Government Data Systems,* Vol. 12, No. 6, November-December, 1983.

[DeLauer] - Richard DeLauer, Under-Secretary of Defense for Research and Engineering, Memorandum on "DoD Policy on Standardization of Host-to-Host Protocols for Data Communications Networks", March, 1982.

[DoDCSC] - *Department of Defense Trusted Computer System Evaluation Criteria,* CSC-STD-001-83, Department of Defense Computer Security Center, Ft. George G. Meade, MD 20755, 15 August 1983.

[Ennis] - Gregory Ennis, "Development of the DoD Protocol Reference Model", *SIGCOMM '83 Symposium: Communications Architectures and Protocols,* March, 1983.

[FACC] - Ford Aerospace & Communications Corporation, *Draft Communications Network Criteria,* Version 1.12, March 1984.

[Gomberg] - David A. Gomberg, William H. Blankertz, George A. Huff and Robert W. Shirey, *WIS Local Area Network System Concepts Paper,* MTR-83W00025, June 1983.

[ISO] - *Open Systems Interconnection - Basic Reference Model,* Draft Proposal 7498, ISO/TC 97/SC 16, August, 1981.

[Postel] - Jon Postel, ed. *Internet Protocol Transition Workbook,* Network Information Center, SRI International, Menlo Park CA 94025 March 1982.

[Rushby] - John Rushby and Brian Randell, "A Distributed Secure System", *IEEE Computer,* July 1983.

AUTOMATED DATA PROCESSING SECURITY ACCREDITATION PROGRAM
(A COMPOSITE GUIDELINE)

JOHN S. COCHRANE SR.
STAFF COMPUTER SYSTEMS ENGINEER
MARTIN MARIETTA, DENVER AEROSPACE

FOREWORD

The data developed within this documentation consists of a sample accreditation program containing composite information extracted from the requirements documents addressing ADP Security for the Department of Defense, U.S. Army, U.S. Navy, U.S. Air Force, and Defense Intelligent Agency.

It is a comparison between various agencies accreditation policies and users will in turn modify as required to apply all or part of this document to their specific needs.

I. Introduction

This document is generic in nature and therefore overviews policies and procedures governing the security accreditation program for Department of Defense (DOD) Automated Data Processing (ADP) facilities. In the context of this document, the term "accreditation" is used to describe the process whereby information pertaining to the security of a Data Processing Installation (DPI) is collected, analyzed, and submitted for a Designated Approving Authority's (DAA) approval. The DAA, upon reviewing the documentation provided via the accreditation program may either concur, thereby certifying that an acceptable level of security is present, or non-concur, indicating that the level of risk has not been reduced to a satisfactory level.

II. Purpose

This document provides generic guidelines to be used in the preparation of an ADP Security Accreditation Program. The objective is to ensure that ADP personnel, and others who may be involved in the planning process, are aware of the information which should be included in such a program, to provide a recommended structure and suggested format, and generally to make those responsible aware of the criticality of the accreditation process.

III. Scope

This document applies to all DPI's and computer systems proposed for the processing of classified information except those listed below:

a. Those preprogrammed and embedded in non-ADP machines or devices and which operate essentially to monitor or control the devices of which they are a part.

b. Programmable calculators and microprocessors used exclusively for mathematical and scientific applications and on which no textual information is processed or stored.

c. Components located within a Communication Center and used exclusively for the functions performed in message switching type communication facilities.

IV. Accreditation Categories

Given the varying types of DOD automated data processing systems, the functions they perform, and the variety of installation housing computer equipment, different accreditation standards and procedures are required. In cases when a DPI has computer systems which operate at different classification levels, procedures and policies will be established for each system according to its classification level. For accreditation purposes, DPI's are grouped according to the following modes of operation: (see Appendix A for definitions)

a. Dedicated
b. System High
c. Controlled
d. Multi-level
e. Compartmented.

No single government document contains all the above modes of operation. The Industrial complex, unless under a specific contract, cannot operate above the Controlled mode of operation. Since this is an overview accreditation program, we have listed all categories that possibly could be used.

V. Reference Guideline Documentation

a.	DOD 5200.28	Security Requirements for Automatic Data Processing (ADP) Systems
b.	DOD 5200.28M	ADP Security Manual . . .
c.	DOD 5220.22M	Industrial Security Manual for Safeguarding Classified Information

d. AFR 300 - 8 Automated Data Processing . . .

e. OPNAVINST 5239.1A Department of Navy Automatic Data Processing Security Program

f. AR 380 - 380 U.S. Army Automated System Security

g. DIAM 50 - 4 Security of Compartmented Computer Operations (Classified Document)

VI. The Accreditation Process

A. Accreditation

Accreditation is the DAA's formal declaration that appropriate ADP Security countermeasures have been properly implemented for the ADP Activity consistent with data protection requirements and that the applicable steps in the accreditation process have been completed. The basic steps of the accreditation process may not vary by level and types of data. However the individual requirements will vary and are presently defined by Department of Defense documentation i.e. DOD 5220.22M, Defense Intelligence Agency, i.e. DIAM 50 - 4, and Military regulations i.e. AFR 300 - 8, together with the DAA's interface and concurrence. The accreditation process requires information gathering, analysis, and formal review by management within both the operating and certifying environments. These accreditation processes must be conducted so as to include the following security categories:

Physical Security
Information Security
Personnel Security
Communication Security
Computer Security

The accreditation process should involve the following essential actions: (additional actions may be added at the direction of the DAA)

1. Statement of DPI goals and objectives; to include a review of security requirements and implementation methods for the operations requiring certification. This can be a separate document i.e. ADP Security Requirements/Implementation Plan or it can be included in the Project Management Plan (PMP), System Requirements Review (SRR), etc.

2. Detailed risk analysis i.e. FIPS 65, to identify vulnerabilities and evaluate risks, and countermeasures available to minimize risks. A risk analysis is required if the DAA is the Defense Intelligence Agency but is usually a requested option by other Designated Approving Authorities.

3. Detailed descriptions of proposed operation i.e., modes, users, security level and services to be provided to include precise definition of security features forming the basis for accreditation. This detailed description will be identified as the ADP Security Plan (reference Appendix B) which requires Designated Approving Authority approval.

4. Coordination of the acquisition and implementation of all the required security features, i.e. Design reviews (PDR), Computer Program Configuration (CPCI), Equipment Configuration Item (CI) etc.

5. A plan for a system security test and evaluation, test and evaluation team, test procedures, etc. This effort will be documented through the development of the ADP Security Test and Evaluation Plan (Reference Appendix C) which will be approved by the Designated Approving Authority. *1

6. Establishment of a Test Team to conduct a security test and evaluation in accordance with the approved Security Test and Evaluation Plan.

7. Statement of continuing problem areas, resources requirements and impact will be developed. The Security Test and Evaluation Report (Reference Appendix D) documents the execution and results of the Security Test and Evaluation Plan. It analyses the findings, and/or vulnerabilities, and lists recommendations for corrections as required.

8. A formal request for accreditaton (Reference Appendix E) with the above supporting documents will be addressed to the Designated Approving Authority.

9. A Contingency Plan is often a Designated Approving Authority required option within the accreditation program. The detail and scope of the plan depends upon the characteristics of the individual activity and should provide detailed procedures for all aspects of emergency backup, and recovery operations.*2

(*1) - Items 5, 6, and 7 are not an accreditation process within the Industrial Security environment.

(*2) - Contingency Planning is only identified in the following Regulations; OPNAVINST 5239.1A, AFR 300-8, DIAM 50-4.

Since the documentation associated with the formal accreditation describes in detail vulnerabilities, risks, and physical layout of the DPI, consideration should be given to classifying such documentation to a level commensurate with the highest classification of information processed. A review of the Project Classification Guide and/or direction from the Designating Approving Authority should assist in determining the classification of these documents.

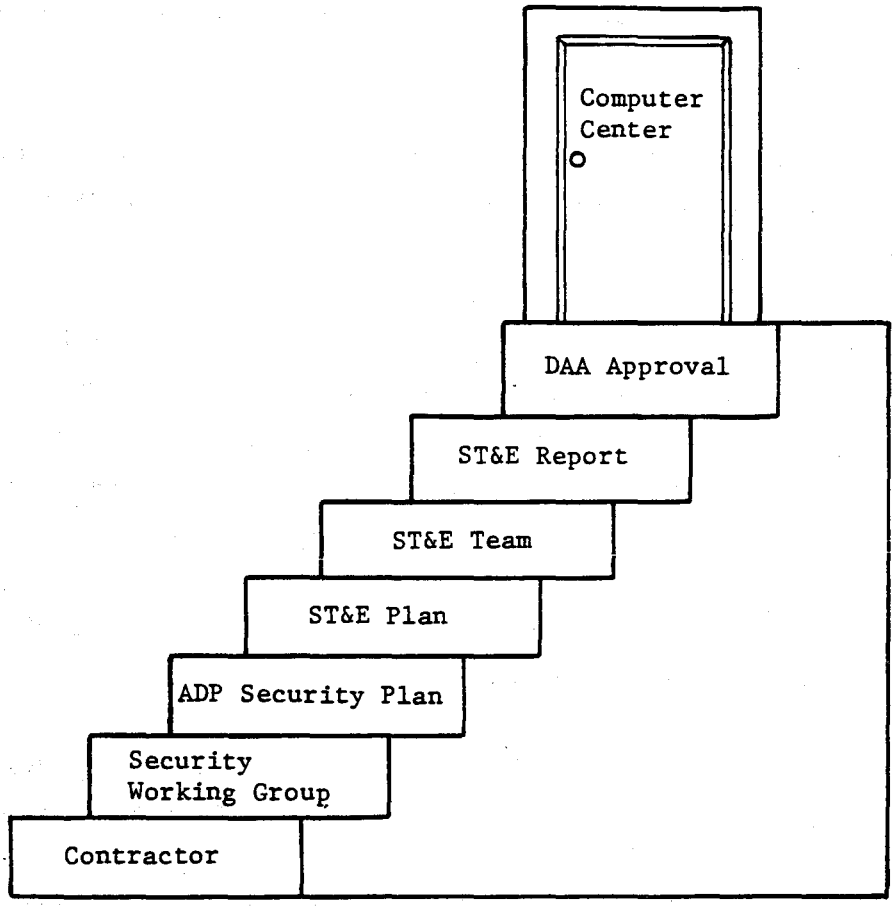


Figure A Accreditation Program

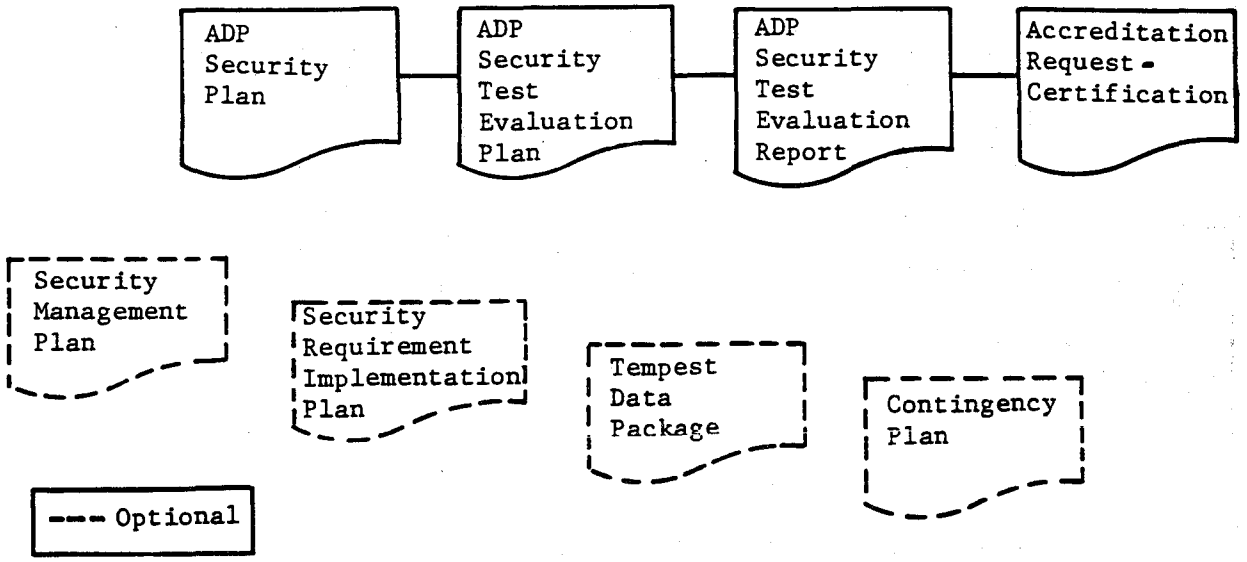


Figure B Accreditation Documentation Flow

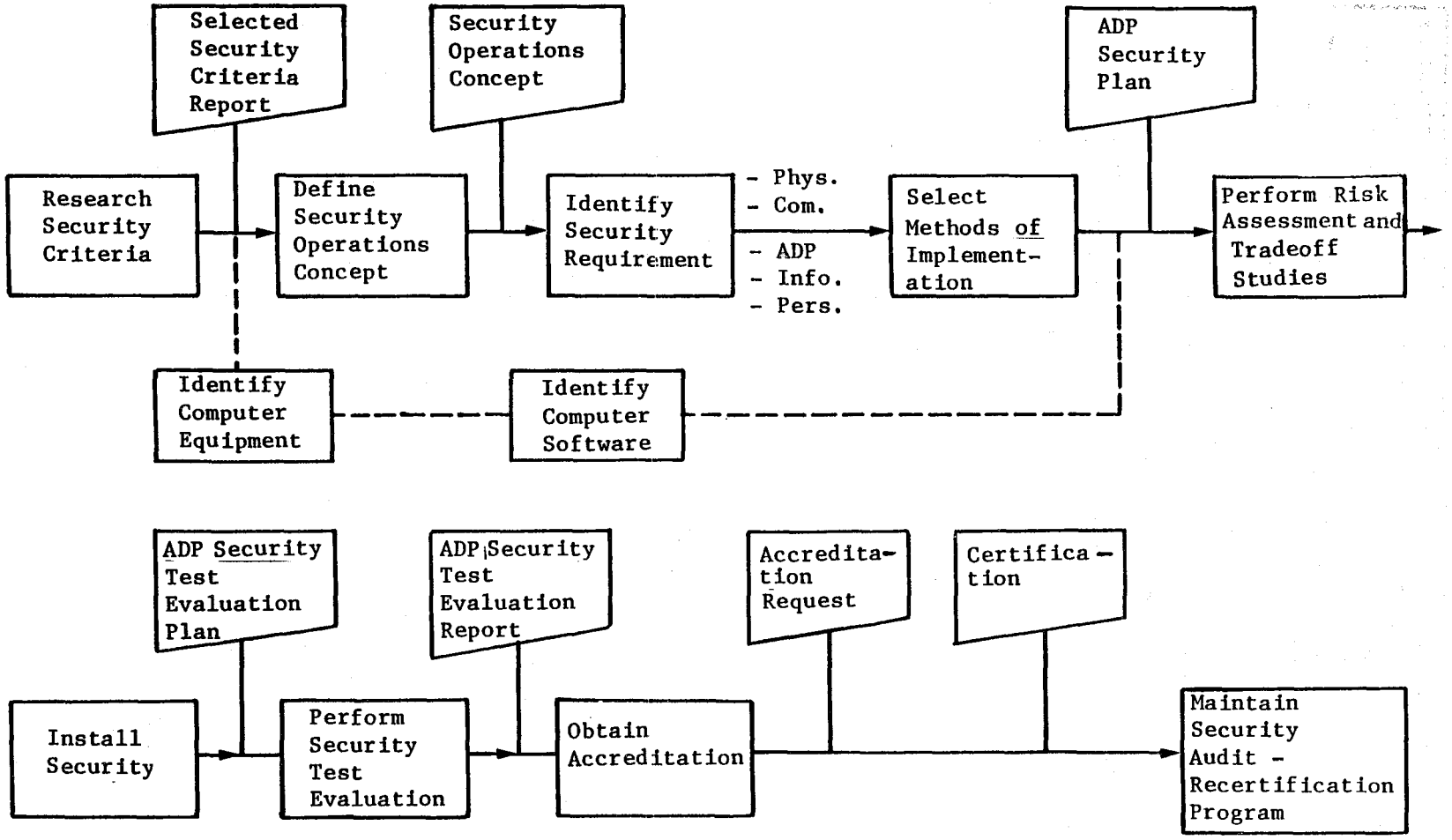


Figure C Accreditation Program Flow

B. Certification

All DPI's designated to process classified data will be certified prior to commencing operations at that classified level. The Designated Approving Authority will issue a Certification Statement (Reference Appendix F). A DPI not accredited may operate with classified data if the appropriate Designated Approving Authority has issued an interim authority to operate. This interim authority to operate while continuing the accreditation process permits the activity to meet critical operational mission requirements while improving its ADP security posture.

Interim authority to operate is not a waiver of the requirement for accreditation.

C. Re-certification

Each DPI will undergo a formal re-certification review according to the schedule assigned by the DAA. However most government regulations recommend that the review be performed as a minimum "annually".

DPI's will also be recertified when the DAA or ADP security staff determines that a change has been made which voids the original accreditation conditions. Significant changes which could impact the ADP security position are:

1. A change in the level or type of data being processed, i.e., secret to top secret or collateral to SCI.

2. Addition or replacement of a mainframe or major system components.

3. A complete revision, new release or update to the operating system.

4. A breach of security, violation of system integrity or unusual situation which appears to invalidate the original condition of certification.

5. Construction or modification of the ADP facility.

6. A change in the mode of operation, i.e., System High to Controlled mode of operation.

VIII. Various Accreditation Tools

Automated Data Processing Systems directed to operate in a security mode of operation of a higher level than System High will be required to demonstrate an enhanced Security Control in Computer Security, thereby requiring an in-depth review of Computer Hardware/Software Security controls. Listed in this section are some of the Computer Security evaluation tools available to exemplify accreditation. (This is not intended to be a total listing.)

A. Security Model

A Security Model is an evaluation tool to which identifies the entities in a computer system are abstractly divided into sets of subjects (active entities, such as processes) and objects (information containers). Within the model the motion of a secure state is defined, and an inductive proof of system security is defined, and an inductive proof of system security can be given. A system state is defined to be "secure" if the only permitted access of subjects to objects are in accordance with specified level restrictions.

B. Trusted Computer System Evaluations

The DOD Computer Security Center has developed a Trusted Computer System evaluation criteria (CSC-STD-001-83 DOD Trusted Computer System Evaluation criteria) which provides a basis for the evaluation of effectivness of security controls built into automatic data processing system products. The criteria has been developed with three objectives in mind: (a) to provide users with a yardstick with which to assess the degree of trust that can be placed in computer systems for the secure processing of classified or other sensitive information, (b) to

provide guidance to manufacturers as to what to build their new, widely available trusted commercial products in order to satisfy trust requirements for sensitive applications, and (c) to provide a basis for specifying security requirements in acquisition specifications.

C. Security Checklist

Each agency has developed a series of ADP Security checklists. These checklists are either a composite or an individual security document which well covers all Security categories, i.e. Physical, TEMPEST, Computer, etc. Reference and/or example of checklists can be identified in the governing security documents for a users given program.

The follow appendixes are available upon request from the DoD Computer Security Center.

Appendix A - Accreditation Categories
Appendix B - The ADP Security Plan
Appendix C - The ADP Security Test and Evaluation Plan
Appendix D - The ADP Security Test and Evaluation Report
Appendix E - System Accreditation Request
Appendix F - System Certification Statement
Appendix G - Bibliography

Please address requests to:

DoD Computer Security Center
9800 Savage Rd.
Fort George G. Meade, MD 20755-6000

Attn: M. Vaughan, C4

CONFIGURATION MANAGEMENT FOR CERTIFIED AND ACCREDITED OPERATIONAL SYSTEMS

LYNNE S. VIDMAR AND LESLEE L. O'DELL

DEPARTMENT OF DEFENSE COMPUTER SECURITY CENTER

INTRODUCTION

The philosophical basis for which this paper has been initiated is to provide well defined Configuration Controls that address security related changes to a certified and accredited operational system. We feel that the purpose of these Configuration Management Procedures is to control the process in which a change is made to a previously certified system in order to reduce the amount of effort to certify this modified system. Discussed in this paper are the reasons for which an operational system may need to be re-certified, controls which must be followed during a change to system security features, and requirements that must be met after the system change has been completed. It is important to define Configuration Controls for certified and accredited operational systems because currently there is not a Configuration Control Plan with respect to the DoD Trusted Computer Evaluation Criteria that addresses changes which affect the code of the Trusted Computing Base (TCB).

We concluded that when a change is performed which affects any code residing in the TCB, it creates a modified or new version of the system. Any change made to the TCB invalidates that system's current certification and accreditation rating. Therefore, for any operational system rated D-A1 in accordance with the Criteria, we suggest that when any change is made to the system's TCB, this new system should be certified. Any changes made to the system's TCB are considered to violate the current assurance of the system's security state. We have defined a security state as the certification rating of the original system. In conclusion, the system's current certification and accreditation is maintained only when there are no changes to the TCB.

FORMAT OF THE PAPER

There are two major divisions in this paper. The first division contains the Configuration Management Controls for systems falling under D-B1 evaluation Classes and the second division contains the Configuration Management Controls for systems falling under B2-A1 evaluation Classes. The reason for the two major divisions is to separate the methods by which the change is made, the change is tested, and the change is evaluated. For systems being changed that fall under the D-B1 Classes, the entire TCB shall be evaluated and tested because their architecture does not support Security Domains of Execution. For example, when a change is made to the TCB of a C1 rated system, the entire system which is the TCB must be tested and certified. Since there is the concept of Security Domains of Execution in the

Classes B2-A1, evaluating and testing is performed differently. This will be more thoroughly explained in the Introduction of the Configuration Management Procedures for the B2-A1 evaluation Classes (P. 11). All configuration controls contained in classes D-B1 will establish the basis for the controls for systems rated B2-A1 class.

For the context of this paper, each of the two divisions of Configuration Management is broken into four parts. Part one is the process by which changes are considered and reviewed. Part two are the Configuration Controls to be followed when making a change. The third part consists of both the testing requirements and the check list. The check list for the system's class is all the requirements for that class as specified in the "DoD Trusted Computer System Evaluation Criteria". The fourth and final part contains the recording and reporting of changes to both the system and procedures which are used in the operational state. An operational state is defined as the condition or mode when a system is declared to be properly implementing its desired functions. We believe each process must be rigidly adhered to, in a distinct order, to ensure an employable configuration control plan for the system. We feel that if any one of these processes fails, the entire Configuration Management process fails.

CONFIGURATION MANAGEMENT FOR D-B1 EVALUATION CLASSES

INTRODUCTION

Configuration Management is necessary for all systems. Systems rated C1-B1 have limited but necessary security features. Therefore, to maintain that these systems continue to uphold their minimum security assurances, we are suggesting that our proposed configuration controls be followed. For a system rated in the D class, we feel a configuration control plan is necessary because a change may upgrade the system to a higher class. For example, in order for a D rated system to be upgraded and certified as a C1 system, the change must follow the Configuration Control process for a C1 system.

THE ROLE OF THE CONFIGURATION BOARD AND MANAGER

Before any changes can be made to the TCB, we suggest a design of the change be brought before a Security Configuration Control Board (SCCB). This Board could be composed of an accreditor, certifier, and a security representative and anyone they deem appropriate in facilitating the change. For example,

Engineers- who would reside on the Board to address hardware or interface changes,

Operations Personnel- who would reside on the Board to address the complications in running the system once the change has been made,

System Software Support- who would reside because they will have to re-compile all changes in the baseline master copy,

Vendor- who would reside on the Board if the change is a vendor change to give assistance in defining the proposed change,

and others in areas such as product assurance, logistic support, production, maintenance, test, procurement, facilities, interface control, users, and others. The accreditor(s) will be the chairman of the SCCB with a designated alternate(s). The chairman, his alternate, and the remaining members will be specifically named on special orders/charters establishing the SCCB. These orders should be revised periodically to reflect appropriate changes in SCCB composition. We feel there should be a document which will contain the concurrence/non-concurrence of the Board as a whole, of each member, the established implementation need date, and the recommended method of implementation. Controls shall be established for the running of the Board. We suggest the Board adhere to the following rules: before a session may begin, the accreditor(s), certifier, and security representative must be in attendance; as for the rest of the Board, more than fifty percent (50%) of the members must be present for the Board to convene. The recommendations given to the accreditor represent not only each members concurrence/non-concurrence, but also the Boards majority rule; therefore, a 50+% attendance requirement.

The following are suggested steps that the Board should perform:

1. The requesting office or vendor should submit their proposed change according to the format given in figure 1 to the accreditor, certifier, and security representative. These SCCB members shall review the information provided in the Proposed Change Request Form (Fig. 1) to determine whether the proposed change is TCB relevant. If the change is decided to be TCB relevant, then the office or vendor requesting the change will submit a formal Security Change Request Form (Fig. 2) and the Board shall convene and begin the process of considering and reviewing the overall impact of the change to the system. The format of the security request change includes a definition of the change, what the change is, impact the change will have on the TCB, risks involved, etc. If the proposed change is not security related, then the change will fall under a different Configuration Control Plan not discussed in this paper.

2. We envision that every TCB change will be critically evaluated, including as an alternative, not making the proposed change. This Board will assess the overall impact that the design change may have on the system's security. Although a change in design, implementation, and its impact on the system's security is the Boards major concern, the following represent other factors to be considered in the Boards decision: the costs and time needed for the modification, possible risks involved, performance reliability, maintainability, schedule, operational effectiveness, safety and human factors, logistic support, and transportability. We suggest a maximum thirty (30) day time limitation on the SCCB for their recommendations to the accreditor. The documented recommendations will be presented to the accreditor for his review.

3. We concluded that the accreditor(s) shall make the final decision to approve or disapprove the implementation of the security related change based on the Boards recommendations. Again, we recommend that the accreditor(s) has a maximum of sixty (60) days for this decision. If the change is allowed then the accreditor(s) shall submit the approved change request to the Configuration Manager. If the proposed security related change is not approved, the accreditor submits the disapproved change and his documented reasons for disapproval to the Configuration Manager for archival. The reason for documenting all decisions is for historical purposes and for fallback procedures. For example, if a change is submitted to the SCCB which has previously been submitted and reviewed, the Board will already have the necessary information and much time and effort will be saved.

4. If a Configuration Manager currently exists for the system, he should ensure the correct implementation of all Configuration Controls provided in this paper. If a Configuration Manager does not exist, one shall be appointed for the system. The Configuration Manager is responsible for the documentation of every phase of each change, including: drawings, parts lists, specifications, test procedures, evaluations, etc.

5. We believe after testing and evaluating is completed, the Configuration Manager will submit to the SCCB the evaluation and test results of the change. These results will be deliberated upon by the SCCB for their final recommendation to the accreditor(s) for approval or disapproval. Again, there is a suggested five day time limitation on the SCCB for their recommendations. The accreditor(s) shall make the final decision of approval or disapproval based on the SCCBs recommendation. The accreditor has a maximum of five days to make a final decision. Upon disapproval/approval the accreditor shall submit a formal letter to the office requesting the change stating reasons for approval/disapproval. If the accreditor disapproves the evaluated, tested, and certified change, he will submit this decision to the Configuration Manager for archival. If the change is approved, the Configuration Manager will be notified to implement the change and update the documentation.

CONFIGURATION CONTROLS FOR D-B1 CLASSES

The following Configuration Controls shall be enforced by the Configuration Manager for any system rated from D to B1 when changes are made to the trusted computing base (TCB). These Controls shall only be followed after the SCCB has approved the change for any system originally certified and accredited in the Class range of D to B1 systems. All new controls are in **bold** type.

1. The security related change must first be approved by the SCCB before any work may begin.

2. All changes will be made to a copy of the baseline master copy of the system rather than the operational system's code because the baseline master copy is stored in a secure manner and has not been subjected to the operational environment thereby, not exposing it to possible subversive code.

3. After the change is completed, the entire trusted computing base (TCB) must be evaluated, tested, and certified. The evaluation, tests, and certification shall demonstrate that the change did not increase the risk of denial of service.

4. The Configuration Manager must create an orderly identification and scheduling system for initial release of design data and expeditious accomplishment of changes once the drawing or specification is released. Major objectives are the following:

a. Assure proposed changes stay within the bounds of the overall project direction which was given by the Security Configuration Control Board.
b. Establish a change introduction point and disposition for existing materials.
c. Assure changes as defined are correct, clear, complete, and understood by responsible personnel.
d. Integrate the changes among all affected operations.

5. The Manager must establish a central location (integrated record system) where the status of existing hardware and software are known; for example,

a. Quantities of hardware in existence.
b. Stage of completion of hardware, firmware and software and their location.
c. Location of hardware, software and firmware being used.

6. All approved security changes will be made at a central depository which maintains the necessary development tools.

7. The Configuration Manager will submit the Configuration Management procedures implemented during the change, the test results, and an evaluation of the impact of the change on the TCB to the accreditor and his Board.

8. Ensure that any software change conforms to the format of the baseline master copy of the software already in place.

9. Once the accreditor approves the tested security related change, the Configuration Manager is responsible for ensuring the correct implementation of the change to the baseline master copy of the code. A tool shall be provided for comparing the newly generated version of code with the baseline master copy to ascertain that only the approved change has been made. After a successful comparison, the new version of code will be promoted to the baseline master copy.

10. Changes to engineering configuration data shall be properly implemented to assure that the change is minimized and that the configuration item will be produced as specified.

11. All individuals who maintain or alter the system shall be under that facility's requirements for personnel security. These individuals shall have a clearance as high as the highest security classification of data residing on the system. (This is to maintain the integrity of the data residing on the system.)

12. Upon final approval of the evaluated, tested, and certified change, the Configuration Manager is responsible for the secure distribution of all copies of the baseline master copy to all field sites.

13. A combination of physical, procedural, and technical (e.g., cryptographic methods) safeguards shall be used to protect from unauthorized modification or destruction the master copy or copies of all material used to generate the TCB.

14. After final testing and evaluation of the security related change, the accreditor(s) must approve or disapprove the change by evaluating the results submitted by the Configuration Manager.

15. There must be assurance of a secure initial state.

16. There must be assurance of a trusted restart.

17. There must be sufficient controls to ensure that the field sites received the deliverable items in the same structure as they were sent.

18. There shall be aperiodic evaluation of master copies of the security related hardware boards, software and firmware at all field sites.

19. All engineering, hardware, software, firmware and configuration data affected by approved changes shall be revised as necessary to describe the change. The change shall also be shown in all related support elements such as manuals, handbooks, support equipment, etc. - to the extent described in the Configuration Controls for that level.

20. All actions related to the submission, analysis, approval, and implementation of changes shall be thoroughly documented and archived.

21. All documentation, source code, object code, compilers, support software, processed information, computers and any peripheral devices shall be maintained in a facility with a Classification level no lower than the highest Classification of any of the items mentioned above.

22. Every changed line of code to the copy of the baseline master copy shall be initialed by the changer and archived. This approved initialed copy of the baseline master copy shall be used as a cross reference against the original baseline master copy when the update is made.

TESTING REQUIREMENTS

The following section establishes testing requirements which must be satisfied for system's rated D-B1 Class. The Configuration Manager is responsible for the correct implementation of all controls during the test phase of the change. He is also responsible for obtaining an appropriate qualified organization to perform the evaluation, testing and certification of the changed system. When the system falls

under one of these Classes, the entire TCB must be evaluated and tested. All the requirements which are stated under a certain Class for that system must be satisfied. Testing requirements contained in a lower Class will also apply to all consecutive hierarchical Classes. All new requirements to each Class are in **bold** type.

D Class Testing Requirements

There are no specific testing requirements at this level. This Class is reserved only for those systems which have been evaluated and tested and do not meet any other higher evaluation Class.

C1 Class Testing Requirements

After changes are made by appropriate personnel, testing shall be performed on the system by a qualified organization. The following test requirements shall apply at the C1 level:

- **The TCB shall be tested and found to work as claimed in the updated system documentation and as defined by the DoD Trusted Computer System Evaluation Criteria.**

- **The tests shall assure that there are no obvious ways for an unauthorized user to bypass or otherwise defeat the security protection mechanisms of the TCB.**

- **An updated test plan which incorporates the security changes shall be used.**

C1 Check List For Testing

The check list for the testing requirements are the requirements found in the "DoD Trusted Computer System Evaluation Criteria" specified under the C1 class. The Configuration Manager is responsible for documenting all controls used during testing.

C2 Class Testing Requirements

After changes are made by appropriate personnel, testing shall be performed on the system by a qualified organization. The following test requirements shall apply at the C2 level:

- The TCB shall be tested and found to work as claimed in the updated system documentation and as defined by the DoD Trusted Computer System Evaluation Criteria.

- The tests shall assure that there are no obvious ways for an unauthorized user to bypass or otherwise

defeat the security protection mechanisms of the TCB.

- An updated test plan which incorporates the system changes shall be used.

- **Testing shall include a search for obvious flaws that would allow violation of resource isolation, or that would permit unauthorized access to the audit or authentication data.**

C2 Check List For Testing

The check list for the testing requirements are the requirements found in the "DoD Trusted Computer System Evaluation Criteria" specified under the C2 class. The Configuration Manager is responsible for documenting all controls used during testing.

B1 Class Testing Requirements

After changes are made by appropriate personnel, testing shall be performed on the system by a qualified organization. The following test requirements apply at the B1 level:

- The TCB shall be tested and found to work as claimed in the updated system documentation and as defined by the DoD Trusted Computer System Evaluation Criteria.

- The tests shall assure that there are no obvious ways for an unauthorized user to bypass or otherwise defeat the security protection mechanisms of the TCB.

- An updated test plan which incorporates the system changes shall be used.

- Testing shall include a search for obvious flaws that would allow violation of resource isolation, or that would permit unauthorized access to the audit or authentication data.

- **The system shall be subject to thorough analysis and testing by a team of individuals who understand the specific implementation of the TCB's design documentation, source code, and object code. This teams' objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users.**

- **All discovered flaws shall be removed or neutralized and the TCB retested to demonstrate that they have been eliminated and that no new flaws have been introduced.**

B1 Check List For Testing

The check list for the testing requirements are the requirements found in the "DoD Trusted Computer System Evaluation Criteria" specified under the B1 class. The Configuration Manager is responsible for documenting all controls used during testing.

REQUIRED DOCUMENTATION

The following section indicates required documentation which must, when applicable, be updated to reflect any tested and approved change to the system for system's rated D-B1 Class. All the requirements which are stated under a certain Class for that system must be satisfied. Required documentation contained in a lower Class will also apply to all consecutive hierarchical Classes. All new requirements to each Class are in **bold** type.

D CLASS DOCUMENTATION

There are no specific documentation requirements at this level. This Class is reserved only for those systems which have been evaluated and tested and do not meet any higher evaluation Class.

C1 CLASS DOCUMENTATION

The following are the specific C1 evaluation class documentation requirements which, when applicable, must reflect the accreditor approved security related change:

1. All test plans and results of the security mechanisms' functional testing shall be documented.

2. There shall be a summary, chapter, or manual in the user documentation which describes the protection mechanisms provided by the TCB, guidelines of their use, and how they interact with one another.

3. There shall be a manual addressed to the ADP system administrator which cautions about functions and privileges that should be controlled when running a secure facility.

4. There shall be a document available which describes and explains the security policy(s) being implemented. If the TCB is composed of distinct modules, the interfaces between these modules shall be described.

C2 CLASS DOCUMENTATION

The following are the specific C2 evaluation class documentation requirements which, when applicable, must reflect the accreditor approved security related change:

1. All test plans and results of the security mechanisms' functional testing shall be documented.

2. There shall be a summary, chapter, or manual in the user documentation which describes the protection mechanisms provided by the TCB, guidelines of their use, and how they interact with one another.

3. There shall be a manual addressed to the ADP system administrator which cautions about functions and privileges that should be controlled when running a secure facility. **The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given.**

4. There shall be a document available which describes and explains the security policy(s) being implemented. If the TCB is composed of distinct modules, the interfaces between these modules shall be described.

B1 CLASS DOCUMENTATION

The following are the specific B1 evaluation class documentation requirements which, when applicable, must reflect the accreditor approved security related change:

1. All test plans and results of the security mechanisms' functional testing shall be documented.

2. There shall be a summary, chapter, or manual in the user documentation which describes the protection mechanisms provided by the TCB, guidelines of their use, and how they interact with one another.

3. There shall be a manual addressed to the ADP system administrator which cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given. **The manual shall describe the operator and administrator functions related to security, to include changing the security characteristics of a user. It shall provide guidelines on the consistent and effective use of the protection features of the system, how they interact, how to securely generate a new TCB. The manual shall also provide facility procedures, warnings, and privileges that need to be controlled in order to operate the facility in a secure manner.**

4. There shall be a document available which describes and explains the security policy(s) being implemented. If the TCB is composed of distinct modules, the interfaces between these modules shall be described. **An informal or formal**

description of the security policy model enforced by the TCB shall be available and an explanation provided to show that it is sufficient to enforce the security policy. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model.

CONFIGURATION MANAGEMENT FOR B2-A1 EVALUATION CLASSES

INTRODUCTION

Unlike the D-B1 Classes where the entire TCB must be evaluated and tested, this division of Classes B2-A1 introduces the new concept of evaluating and testing only particular Security Domains of Execution. A Security Domain of Execution is an architectural structure within the TCB which supports only distinct modules of security code. Security Domains of Execution provide a distinct separation between non-protection critical elements and protection critical elements. The purpose of these Domains is to reduce the effort required to certify a changed system. Before a security related change can be made to the TCB, all the Domains which are directly or indirectly affected by the change must be presented to the Security Configuration Control Board (SCCB). Before the change is made, there shall be an analysis conducted by the office or vendor requesting the TCB change to identify those affected Domains to the SCCB. After the TCB change has been made to a copy of the baseline master copy of the code, the appropriate qualified organization shall perform an analysis to determine which Domains were actually affected by the TCB change. All established affected Domains will be evaluated and tested. However, during the evaluating or testing of these affected Domains, a Domain not previously stated may be found. If this occurs, then this Domain must undergo the same rigorous evaluating and testing as the affected Domains. All other Domains that were not directly or indirectly affected by the change do not need to be evaluated and tested because they were not affected by the change. The affected Domains which are evaluated and tested must fulfill all the requirements (which that Domain implements) specified in the TESTING REQUIREMENTS check list for that system's particular Class. When checking a change which directly or indirectly affects a Domain, the Domain must satisfy all the regulations under that specific feature(s).

THE ROLE OF THE CONFIGURATION BOARD AND MANAGER

Before any changes can be made to the TCB, a design of the change will be brought before a Security Configuration Control Board (SCCB). Again, it is suggested that this Board will be composed of an accreditor, certifier, and a security representative and anyone they deem appropriate in facilitating the change. For example,

Engineers- who would reside on the Board to address hardware or interface changes,

Operations Personnel- who would reside on the Board to address the complications in running the system once the change has been made,

System Software Support- who would reside because they will have to re-compile all changes in the baseline master copy,

Vendor- who would reside on the Board if the change is a vendor change to give assistance in defining the proposed change,

and others in areas such as product assurance, logistic support, production, maintenance, test, procurement, facilities, interface control, users, and others. The accreditor(s) will be the chairman of the SCCB with a designated alternate(s). The chairman, his alternate, and the remaining members will be specifically named on special orders/charters establishing the SCCB. These orders should be revised periodically to reflect appropriate changes in SCCB composition. There shall be a document which will contain the concurrence/non-concurrence of the Board as a whole, of each member, the established implementation need date, and the recommended contractual method of implementation. Controls should be established for the running of the Board. It is suggested that this Board adhere to the following rules: before a session may begin, the accreditor(s), certifier, and security representative must be in attendance; as for the rest of the Board, more than fifty percent (50%) of the members must be present for the Board to convene. The recommendations given to the accreditor represent not only each members concurrence/non-concurrence, but also the Boards majority rule; therefore, a 50+% attendance requirement.

The following are suggested steps that the Board should perform:

1. The requesting office or vendor should submit their proposed change according to the format given in figure 1 to the accreditor, certifier, and security representative. These SCCB members shall review the information provided in the Proposed Change Request Form (Fig. 1) to determine whether the proposed change is TCB relevant. If the change is decided to be TCB relevant, then the office or vendor requesting the change will submit a formal Security Change Request Form (Fig. 3). Then, the Board shall convene and begin the process of considering and reviewing the overall impact of the change to the system. The format of the security request change will include a definition of the change, what the change is, impact the change will have on the TCB, risks involved, etc. Also included in this specification, will be a detailed description of the Security Domains of Execution that are indirectly or directly affected by the security related change. If the proposed change is not security related, then the change will fall under a different Configuration Control Plan not discussed in this paper.

2. Every TCB change will be critically evaluated, including as an alternative, not making the proposed change. This Board will assess the overall impact that the design change may have on the system's security. Although a change in design and its impact on the system's security is the Boards major concern, the following represent other factors to be considered in the Boards decision: the costs and time needed for the modification, possible risks involved, performance reliability, maintainability, schedule, operational effectiveness, safety and human factors, logistic support, and transportability. It is suggested that there be a maximum thirty (30) day time limitation on the SCCB for their recommendations to the accreditor. The documented recommendations will be presented to the accreditor for his review.

3. The accreditor(s) shall make the final decision to approve or disapprove the implementation of the security related change based on the Boards recommendations. Again, it is suggested that the accreditor(s) have a maximum of sixty (60) days for this decision. If the change is allowed then the accreditor(s) shall submit the approved change request to the Configuration Manager. If the proposed security related change is not approved, the accreditor submits the disapproved change and his documented reasons for disapproval to the Configuration Manager for archival. The reason for documenting all decisions is for historical purposes and for fallback procedures. For example, if a change is submitted to the SCCB which has previously been submitted and reviewed, the Board will already have the necessary information and much time and effort will be saved.

4. If a Configuration Manager currently exists for the system, he will ensure the correct implementation of all Configuration Controls provided in this paper. If a Configuration Manager does not exist, one shall be appointed for the system. The Configuration Manager is responsible for the documentation of every phase of each change, including: drawings, parts lists, specifications, test procedures, evaluations, etc.

5. After the testing and evaluating is completed, the Configuration Manager will submit to the SCCB the evaluation and test results of the change. These results will be deliberated upon by the SCCB for their final recommendation to the accreditor(s) for approval or disapproval. Again, there shall be a maximum five day time limitation on the SCCB for their recommendations. The accreditor(s) shall make the final decision of approval or disapproval based on the SCCBs recommendation. The accreditor has a maximum of five days to make a final decision. Upon disapproval/approval the accreditor shall submit a formal letter to the office requesting the change stating reasons for approval/disapproval. If the accreditor disapproves the evaluated, tested, and certified change, he will submit this decision to the Configuration Manager for archival. If the change is approved, the Configuration Manager will be notified to implement the change and update the documentation.

CONFIGURATION CONTROLS FOR B2-A1 CLASSES

The following Configuration Controls shall be enforced by the Configuration Manager for systems which have been accredited and certified B2 through A1. These Controls shall only be followed after the SCCB has approved the change for any system originally certified and accredited in the Class range of B2 to A1 systems. All new controls are in **bold** type.

1. The security related change must first be approved by the SCCB before any work may begin.

2. All changes will be made to a copy of the baseline master copy of the system rather than the operational system's code because the baseline master copy is stored in a secure manner and has not been subjected to the operational environment thereby, not exposing it to possible subversive code.

3. **After the change is completed, only the directly or indirectly involved Security Domains of Execution of the trusted computing base (TCB) must be evaluated and tested. The evaluating and testing shall demonstrate that the change did not increase the risk of denial of service.**

4. The Configuration Manager must create an orderly identification and scheduling system for initial release of design data and expeditious accomplishment of changes once the drawing or specification is released. Major objectives are the following:

a. Assure proposed changes stay within the bounds of the overall project direction which was given by the Security Configuration Control Board.
b. Establish a change introduction point and disposition for existing materials.
c. Assure changes as defined are correct, clear, complete, and understood by responsible personnel.
d. Integrate the changes among all affected operations.

5. The Manager must establish a central location (integrated record system) where the status of existing hardware and software are known; for example,

a. Quantities of hardware in existence.
b. Stage of completion of hardware, firmware and software and their location.
c. Location of hardware, software and firmware being used.

6. All approved security changes will be made at a central depository which maintains the necessary development tools.

7. The Configuration Manager will submit the Configuration Management procedures implemented during the change, **the Security Domains of Execution affected,** the test results, and an evaluation of the impact of the change to the accreditor and his Board.

8. Ensure that any software change conforms to the format of the baseline master copy of the software already in place.

9. Once the accreditor approves the tested security related change, the Configuration Manager is responsible for ensuring the correct implementation of the change to the baseline master copy of the code. A tool shall be provided for comparing the newly generated version of code with the baseline master copy to ascertain that only the approved change has been made. After a successful comparison, the new version of code will be promoted to the baseline master copy.

10. Changes to engineering configuration data shall be properly implemented to assure that the change is minimized and that the configuration item will be produced as specified.

11. All individuals who maintain or alter the system shall be under that facility's requirements for personnel security. These individuals shall have a clearance as high as the highest security classification of data residing on the system. (This is to maintain the integrity of the data residing on the system.)

12. Upon final approval of the evaluated and tested change, the Configuration Manager is responsible for the secure distribution of all copies of the baseline master copy to all field sites.

13. A combination of physical, procedural, and technical (e.g., cryptographic methods) safeguards shall be used to protect from unauthorized modification or destruction the master copy or copies of all material used to generate the TCB.

14. After final testing and evaluation of the security related change, the SCCB must approve or disapprove the outcome of the results. **All effected Security Domains of Execution must be well documented.**

15. There must be assurance of a secure initial state.

16. There must be assurance of a trusted restart.

17. There must be sufficient proof that the field sites received the deliverable items in the same structure as they were sent.

18. There shall be aperiodic re-distribution of master copies of the security related hardware boards, software and firmware to all field sites.

19. All engineering, hardware, software, firmware and configuration data affected by approved changes shall be revised as necessary to describe the change. The change shall also be shown in all related support elements such as manuals, handbooks, support equipment, etc. - to the extent described in the Configuration Controls for that level.

20. All actions related to the submission, analysis, approval, and implementation of changes shall be thoroughly documented.

21. All documentation, source code, object code, compilers, support software, processed information, computers and any peripheral devices shall be maintained in a facility with a Classification level no lower than the highest Classification of any of the items mentioned above.

22. The Configuration Manager shall ensure that the TCB maintain a certifiable architectural structure after a change is completed (i.e., layering, abstraction, data hiding...).

23. The Configuration Manager shall ensure that the TCB be internally structured into well-defined largely independent modules. It shall make effective use of available hardware to separate those elements that are protection-critical from those that are not.

24. The Configuration Manager shall ensure that the TCB modules are designed such that the principle of least privilege is enforced. Features in hardware, such as segmentation, shall be used to support logically distinct storage objects with separate attributes (namely: readable, writeable).

25. There shall be assurance of a consistent mapping among all documentation and code associated with the current (updated) version of the TCB. Tools shall be provided for generation of a new version of the TCB from source code. Also available shall be tools, maintained under strict configuration control, for comparing a newly generated version with the previous TCB version in order to ascertain that only the intended changes have been made in the code that will actually be used as the new version of the TCB.

26. The Configuration Manager shall ensure that the TCB support separate operator and administrator functions.

27. Every changed line of code to the copy of the baseline master copy shall be initialed by the changer and archived. This approved initialed copy of the baseline master copy shall be used as a cross reference against the original baseline master copy when the update is made.

28. The Configuration Manager shall ensure that the evaluating and testing organization analyzes the change to the copy of the baseline master copy in order to locate the Security Domains of Execution which were effected by the change. All effected Domains shall be evaluated and tested by this organization.

TESTING REQUIREMENTS

The following section establishes testing requirements which must be satisfied for systems rated B2-A1 Class. When the system falls under one of these Classes, the quantity of testing and evaluating of the TCB will be determined by the Security Domain(s) of Execution which are affected by the change. The Configuration Manager is responsible for documenting all controls used during testing. He is also responsible for obtaining an appropriate qualified organization to perform the evaluation, testing and certification of the changed system. Only those Security Domain(s) of Execution which were affected by the change will be tested and evaluated to those specific features which the Domain(s) enforces. All the requirements contained in a lower Class will also apply to all consecutive hierarchical Classes. All new requirements to each Class are in **bold** type.

B2 Class Testing Requirements

After changes are made by qualified personnel, testing shall be performed on the system by a qualified organization. The following test requirements shall apply to the domains effected by the change at the B2 level:

- The system shall be tested and found to work as claimed in the updated system documentation and as defined by the DoD Trusted Computer System Evaluation Criteria.

- The tests shall assure that there are no obvious ways for an unauthorized user to bypass or otherwise defeat the security protection mechanism of the TCB.

- The tests should be no more stringent or thorough then those from the original certification.

- An updated test plan which incorporates the system changes shall be used.

- Testing shall include a search for obvious flaws that would allow violation of resource isolation, or that would permit unauthorized access to the audit or authentication data.

- The system shall be subject to thorough analysis and testing by a team of individuals who understand the specific implementation of the TCB's design documentation, source code, and object code. This teams' objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users.

- Any change shall cause the TCB to be retested to demonstrate that all discovered flaws have been eliminated and that no new flaws have been introduced.

- **The system changer shall conduct a thorough search for covert storage channels and make a determination (either by actual measurement or by engineering estimation) the maximum bandwidth of each identified channel.**

- **The system testers and changer shall maintain that the TCB be found relatively resistant to penetration.**

- **The system changer shall demonstrate that the TCB implementation is consistent with the updated DTLS.**

B2 Check List For Testing

The check list for the testing requirements are the requirements found in the "DoD Trusted Computer System Evaluation Criteria" specified under the B2 class. The Configuration Manager is responsible for documenting all controls used during testing.

B3 Class Testing Requirements

After changes are made by qualified personnel, testing shall be performed on the system by a qualified organization. The following test requirements shall apply to the domains effected by the change at the B3 level:

- The system shall be tested and found to work as claimed in the updated system documentation and as defined by the DoD Trusted Computer System Evaluation Criteria.

- The tests shall assure that there are no obvious ways for an unauthorized user to bypass or otherwise defeat the security protection mechanism of the TCB.

- The tests should be no more stringent or thorough then those from the original certification.

- An updated test plan which incorporates the system changes shall be used.

- Testing shall include a search for obvious flaws that would allow violation of resource isolation, or that would permit unauthorized access to the audit or authentication data.

- All discovered flaws shall be removed or neutralized and the TCB retested to demonstrate that they have been eliminated and that no new flaws have been introduced.

- The system shall be subject to thorough analysis and testing by a team of individuals who understand the specific implementation of the TCB's design documentation, source code, and object code. This teams' objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users.

- The system developer shall conduct a thorough search for **covert channels** and make a determination (either by actual measurement or by engineering estimation) the maximum bandwidth of each identified channel.

- The system shall maintain that the TCB be **found resistant to** penetration.

- The system shall demonstrate that the TCB implementation is consistent with the updated DTLS.

- There shall be no design flaws and no more than a few correctable implementation flaws found during testing and there shall be reasonable confidence that few remain.

- **A convincing argument shall be given that the updated DTLS is consistent with the model.**

B3 Check List For Testing

The check list for the testing requirements are the requirements found in the "DoD Trusted Computer System Evaluation Criteria" specified under the B3 class. The Configuration Manager is responsible for documenting all controls used during testing.

A1 Class Testing Requirements

After changes are made by qualified personnel, testing shall be performed on the system by a qualified organization. The following test requirements shall apply to the domains effected by the change at the A1 level:

- The system shall be tested and found to work as claimed in the updated system documentation and as defined by the DoD Trusted Computer System Evaluation Criteria.

- The tests shall assure that there are no obvious ways for an unauthorized user to bypass or otherwise defeat the security protection mechanism of the TCB.

- An updated test plan which incorporates the system changes shall be used.

- Testing shall include a search for obvious flaws that would allow violation of resource isolation, or that would permit unauthorized access to the audit or authentication data.

- All discovered flaws shall be removed or neutralized and the TCB retested to demonstrate that they have been eliminated and that no new flaws have been introduced.

- The system shall be subject to thorough analysis and testing by a team of individuals who understand the specific implementation of the TCB's design documentation, source code, and object code. This teams' objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users.

- The system developer shall conduct a thorough search for covert channels and make a determination (either by actual measurement or by engineering estimation) of the maximum bandwidth of each identified channel. **Formal methods shall be used in the analysis.**

- The system shall verify that the TCB remain resistant to penetration.

- The system shall demonstrate that the TCB implementation is consistent with the updated DTLS.

- Testing shall demonstrate that the TCB implementation is consistent with the updated **FTLS.**

- **Manual or other mapping of the FTLS to the source code may form a basis for penetration testing.**

- There shall be no design flaws and no more than a few correctable implementation flaws found during testing and there shall be reasonable confidence that few remain.

- A convincing argument shall be given that the updated DTLS is consistent with the model **and a combination of formal and informal techniques shall be used to show that the updated FTLS is consistent with the model. This verification evidence shall be consistent with that provided within the state-of-the-art of the particular Computer Security Center-endorsed formal specification and verification system used. Manual or other mapping of the updated FTLS to the TCB source code shall be performed to provide evidence of correct implementation.**

Al Check List For Testing

The check list for the testing requirements are the requirements found in the "DoD Trusted Computer System Evaluation Criteria" specified under the Al class. The Configuration Manager is responsible for documenting all controls used during testing.

DOCUMENTATION FOR B2-Al CLASSES

The following section indicates required documentation which must, when applicable, be updated to reflect any tested and approved change to the system for system's rated B2-Al Class. All the requirements which are stated under a certain Class for that system must be satisfied. Required documentation contained in a lower Class will also apply to all consecutive hierarchical Classes. All new requirements to each Class are in **bold** type.

B2 Class Documentation

The following are the specific B2 evaluation class documentation requirements which, when applicable, must reflect the accreditor approved security related change:

1. All test plans and results of the security mechanisms' functional testing shall be documented. **It shall include results of testing the effectiveness of the methods used to reduce covert channel bandwidths.**

2. There shall be a summary, chapter, or manual in the user documentation which describes the protection mechanisms provided by the TCB, guidelines on their use, and how they interact with one another.

3. There shall be a manual addressed to the ADP system administrator which cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given. The manual shall describe the operator and administrator functions related to security, to include changing the security characteristics of a user. It shall provide guidelines on the consistent and effective use of the protection features of the system, how they interact, how to securely generate a new TCB. The manual shall also provide facility procedures, warnings and privileges that need to be controlled in order to operate the facility in a secure manner. **The TCB modules that contain the reference validation mechanism shall be identified. The procedures for secure generation of a new TCB from source after modification of any modules in the TCB shall be described.**

4. There shall be available a document which describes and explains the protection security policy being implemented. **The document shall also describe the interfaces between the TCB modules. A formal description of the security policy model enforced by the TCB shall be available and proven that it is sufficient to enforce the security policy. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model. The descriptive top-level specification (DTLS) shall be shown to be an accurate description of the TCB interface. Documentation shall describe how the TCB implements the reference monitor concept and give an explanation why it is tamperproof, cannot be bypassed, and is correctly implemented. Documentation shall describe how the TCB is structured to facilitate testing and to enforce least privilege. This documentation shall also present the results of the covert channel analysis and the tradeoffs involved in restricting the channels. All auditable events that may be used in the exploitation of known covert storage channels shall be identified. The bandwidths of covert storage channels, the use of which is not detectable by the auditing mechanisms, shall be provided.**

B3 Class Documentation

The following are the specific B3 evaluation class documentation requirements which, when applicable, must reflect the accreditor approved security related change:

1. All test plans and results of the security mechanisms' functional testing shall be documented. It shall include results of testing the effectiveness of the methods used to reduce covert channel bandwidths.

2. There shall be a summary, chapter, or manual in the user documentation which describes the protection mechanisms provided by the TCB, guidelines on their use, and how they interact with one another.

3. There shall be a manual addressed to the ADP system administrator which cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given. The manual shall describe the operator and administrator functions related to security, to include changing the security characteristics of a user. It shall provide guidelines on the consistent and effective use of the protection features of the system, how they interact, how to securely generate a new TCB. The manual shall also provide facility procedures, warnings and privileges that need to be controlled in order to operate the facility in a secure manner. The TCB modules that contain the reference validation mechanism shall be identified. The procedures for secure generation of a new TCB from source after modification of any modules in the TCB shall be described.

4. There shall be available a document which describes and explains the protection security policy being implemented. The document shall also describe the interfaces between the TCB modules. A formal description of the security policy model enforced by the TCB shall be available and proven that it is sufficient to enforce the security policy. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model. The descriptive top-level specification (DTLS) shall be shown to be an accurate description of the TCB interface. Documentation shall describe how the TCB implements the reference monitor concept and give an explanation why it is tamperproof, cannot be bypassed, and is correctly implemented. **The TCB implementation (i.e., in hardware, firmware, and software) shall be informally shown to be consistent with the DTLS. The elements of the DTLS shall be shown, using informal techniques, to correspond to the elements of the TCB.** Documentation shall describe how the TCB is structured to facilitate testing and to enforce least privilege. This documentation shall also present the results of the covert channel analysis and the tradeoffs involved in restricting the channels. All auditable events that may be used in the exploitation of known covert storage channels shall be identified. The bandwidths of covert storage channels, the use of which is not detectable by the auditing mechanisms, shall be provided.

A1 Class Documentation

The following are the specific A1 evaluation class documentation requirements which, when applicable, must reflect the accreditor approved security related change:

1. All test plans and results of the security mechanisms' functional testing shall be documented. It shall include results of testing the effectiveness of the methods used to reduce covert channel bandwidths.

2. There shall be a summary, chapter, or manual in the user documentation which describes the protection mechanisms provided by the TCB, guidelines on their use, and how they interact with one another.

3. There shall be a manual addressed to the ADP system administrator which cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as

well as the detailed audit record structure for each type of audit event shall be given. The manual shall describe the operator and administrator functions related to security, to include changing the security characteristics of a user. It shall provide guidelines on the consistent and effective use of the protection features of the system, how they interact, how to securely generate a new TCB. The manual shall also provide facility procedures, warnings and privileges that need to be controlled in order to operate the facility in a secure manner. The TCB modules that contain the reference validation mechanism shall be identified. The procedures for secure generation of a new TCB from source after modification of any modules in the TCB shall be described.

4. There shall be available a document which describes and explains the protection security policy being implemented. The document shall also describe the interfaces between the TCB modules. A formal description of the security policy model enforced by the TCB shall be available and proven that it is sufficient to enforce the security policy. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model. The descriptive top-level specification (DTLS) shall be shown to be an accurate description of the TCB interface. Documentation shall describe how the TCB implements the reference monitor concept and give an explanation why it is tamperproof, cannot be bypassed, and is correctly implemented. The TCB implementation (i.e., in hardware, firmware, and software) shall be informally shown to be consistent with the **formal top-level specification (FTLS).** The elements of the **FTLS** shall be shown, using informal techniques, to correspond to the elements of the TCB. Documentation shall describe how the TCB is structured to facilitate testing and to enforce least privilege. This documentation shall also present the results of the covert channel analysis and the tradeoffs involved in restricting the channels. All auditable events that may be used in the exploitation of known covert storage channels shall be identified. The bandwidths of covert storage channels, the use of which is not detectable by the auditing mechanisms, shall be provided. **The use and capabilities of hardware, firmware, and software mechanisms not dealt with the FTLS but strictly internal to the TCB (e.g., mapping register, direct memory access I/O) shall be clearly described.**

CONCLUSION

Configuration Controls for Certified and Accredited Operational Systems are necessary to guide the process of changing the TCB. We feel it is vital that these Configuration Control Procedures interact so that the change can be tightly controlled throughout the change process. When changing the TCB in any system rated D-A1, we concluded that the current certified and accredited rating became invalid. But by following our proposed Configuration Controls, the effort in certifying this modified system is reduced especially for systems in the B2-A1 class. The reason for the reduced effort for systems B2-A1 is because of the architectural difference between D-B1 systems and B2-A1 systems. When changing a system rated D-B1, we concluded that the entire TCB must be re-certified because the TCB is effectively the whole system. But because of Security Domains of Execution in systems rated B2-A1, only those affected Security Domains of Execution have to be evaluated and tested towards the system's new certification. Therefore systems in a higher class require less overall, but just as stringent evaluation and testing as systems rated D-B1.

We concluded that the Controls placed on the SCCB are generic for both divisions. This is because the procedures which the SCCB and the Configuration Manager must follow in the process of changing the system are similar for all classes of systems. By having Configuration Controls which closely regulate the change process, the operational site or vendor will be able to change their system in a timely manner.

It must be recognized that this paper is a living document and any suggestions and comments regarding this subject would be greatly appreciated.

ACKNOWLEDGEMENTS

The authors would like to thank Warren Shadle and Don Yeskey for raising this important issue of Configuration Management for Operational Systems and for their guidance and support in the writing of this paper.

REFERENCES

Bersoff, Edward H., Software Configuration Management, An Investment in Product Integrity, 1980.

Department of Defense Joint Services and Agencies, Configuration Management Regulation, AR 70-37, NAVMATINST 4130.1B, MCO 4130.1B, AFR 65-3, DLAR 8250.4, NSA/CSS REG 80-14, DCAC 100-50-2, DNA INST 5010.18, National Security Agency, Ft. Meade, MD, 30 October 1981.

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

Samaras, Thomas T., Fundamentals of Configuration Management, 1971.

Schaefer, M. and R. R. Schell, Toward an Understanding of Extensible Architectures for Evaluated Trusted Computer System Products. IEEE Proceedings of the 1984 Symposium on Security and Privacy, pgs 41-52.

Figure 1

PCR#_____

PROPOSED CHANGE REQUEST - SECURITY REVIEW

Date:__/__/__

Originator:_____ Project Manager:_____

System:_____ System Configuration Manager:_____

CPCI No.:_____ CPC No.:_____

Brief Change Description:

Reason for Change:

_____ _____

Signature of Originator Signature of Project Manager

To Be Completed By Security Review Team

Date:__/__/__

Accreditor:_____ Security Representative:_____

Security Relevant Change: ____ YES ____ NO

_____ _____

Signature of Accreditor Signature of Security Representative

Figure 2

SCR#_____

SECURITY CHANGE REQUEST

Date:__/__/__

Current Evaluation Class: (check one)

___ D ___ C1 ___C2 ___B1

Originator:_____ Project Manager:_____

System:_____ System Configuration Manager:_____

CPCI No.:_____ CPC No.:_____

Reason for Change:

Detailed Change Description:

Impact of Change to System Security:

Proposed Schedule:

Man Months Required:_____ Cost of Change:_____

Requirement for Contractor: ___ YES ___NO

_____ _____

Signature of Originator Signature of Project Manager

To Be Completed By Accreditor

Date:__/__/__ Accreditor:_____

Accreditors Recommendation: ___ APPROVED ___ DISAPPROVED

_____

Signature of Accreditor

Figure 3

SCR#_____

SECURITY CHANGE REQUEST

Date:__/__/__

Current Evaluation Class: (check one)

___ B2 ___ B3 ___ A1

Originator:_____ Project Manager:_____

System:_____ System Configuration Manager:_____

CPCI No.:_____ CPC No.:_____

Reason for Change:

Detailed Change Description:

Impact of Change to System Security:

Affected Security Execution Domains:

Proposed Schedule:

Man Months Required:_____ Cost of Change:_____

Requirement for Contractor: ___ YES ___NO

_____ _____

Signature of Originator Signature of Project Manager

To Be Completed By Accreditor

Date:__/__/__ Accreditor:_____

Accreditors Recommendation: ___ APPROVED ___ DISAPPROVED

_____

Signature of Accreditor

ANALYSIS OF A KERNEL VERIFICATION

Terry C. Vickers Benzel

The Mitre Corporation
Bedford, MA

Abstract

This paper [1] reports on the analysis and evaluation of the SCOMP kernel verification. The SCOMP system was developed by Honeywell FSD and is targeted at the A1 class of the DoD Trusted Computer System Evaluation Criteria [CSC83]. It is currently under evaluation by the Department of Defense Computer Security Center (DoDCSC). The work reported on here is significant in that the SCOMP system is the first commercially-available formally verified operating system. Furthermore, it is the first system to be evaluated against the A1 requirements for formal design specification and verification. The methods and procedures used for this analysis and evaluation will be of interest to future system designers, verifiers, and evaluators. The results of the verification, in particular the types of assurances that were gained, will also be discussed.

I. Introduction

This paper reports on the analysis and evaluation of the SCOMP kernel verification. The SCOMP system was developed by Honeywell FSD and is targeted at the A1 class of the Trusted Computer System Evaluation Criteria [CSC83]. It is currently under evaluation by the Department of Defense Computer Security Center (DoDCSC). The work reported on here is significant in that the SCOMP system is the first commercially-available formally verified operating system. Furthermore, it is the first system to be evaluated against the A1 requirements for formal design specification and verification. The methods and procedures used for this analysis and evaluation will be of interest to future system designers, verifiers, and evaluators. The results of the verification, in particular the types of assurances that were gained, will also be discussed.

The background and historical perspective of the SCOMP verification effort will be presented briefly in Section 2. Section 3 will discuss the formal specification effort, including such concerns as specification for design vs. specification for verification, specification of user-visible vs. user-invokable functions, and detail vs. abstraction. Section 4 will present the verification effort that took place on three occasions during a three-year period. Section 5 will present the results obtained for each of these verification efforts. The covert channel analysis and the methods developed to perform this analysis will be covered in Section 6; of particular interest is the number of covert channels that were found through the use of formal methods as compared to those found through informal methods. Section 7 will discuss the methods and procedures developed to establish the specification-to-code correlation, and the results obtained. Finally, Section 8 will address the types of assurances, security security-related and non-security-related, which were gained as a result of the kernel verification.

II. Background and Historical Perspective

This section will present a brief overview of the historical motivation, verification goals, and an introduction to the HDM verification methodology.

The SCOMP system consists of a commercial Honeywell Level 6 minicomputer enhanced by a hardware Security Protection Module (SPM), security kernel software and a SCOMP system Kernel Interface Package (SKIP). Honeywell worked extensively with various DoD organizations during the design and development of the SCOMP system. Formal methods of specification and verification were employed during the development of the system. One goal of this project was to develop a system which could comply with the DoD policy on automatic processing of multiple levels of classified data [DOD5200] In order to meet this goal Honeywell used the concepts of a reference monitor, security kernel and formal specification and verification. The design of the SCOMP system and its architecture and security mechanisms are discussed in [FRAIM83], and [BENZ83].

The design of the SCOMP security kernel was formally specified and verified using the SRI International Hierarchical Development Methodology (HDM) and tools. The basic concepts of HDM are discussed in [ROBI79]. The design of the SCOMP system kernel was formally specified in the HDM specification language SPECIAL. Then the Multilevel Security tool (MLS) and theorem prover of HDM were used to verify that the kernel obeyed the required security properties. The security properties of interest were derived from the Bell and LaPadula model. [BELL74]. These security properties are defined by Bell and LaPadula in terms of subjects and objects, they are:

- Simple Security: A subject can read a data object only if the subject has a security level greater than or equal to that of the object;
- *-Property: A subject can write a data object only if the subject has a security level less than or equal to that of the object.

The MLS tool is based on the concept of flow analysis, it determines the paths through which information can flow in the specifications. For each path the tool generates a formula which describes the information flow between the variables in the system. The tool then attempts to resolve any formulas which are trivially true by examining the security levels of the variables. Trivially true formulas involve flows of information between two variables at the same level, flows from a variable at system low to any variable, and flows from a variable at any level to a variable at system high. All formulas which the MLS tool is unable to resolve are passed to the Boyer-Moore theorem to be proved. In theory, all formulas which the theorem prover is unable to prove denote invalid (or insecure) information flows in the system. Each unproven formula must be inspected manually in order to determine which flows are truly invalid. The MLS tool and its application to proving design specifications secure is discussed in [FEIER80]. Thus, the primary goal in the SCOMP verification effort was to demonstrate that the design specifications for the SCOMP security kernel obeyed the multilevel security properties.

III. Analysis of the Specifications

The formal specifications for the SCOMP system originally were written at a time when the specification language and tools were still under development. This resulted in numerous revisions to the specifications in order to make them compatible with the developing verification tools. The first version of the formal specifications was written as a design specification and did not include the properties necessary to construct a proof of the system's security. In practice, writing a formal design specification is very different from writing a formal specification which can be subjected to automated verification techniques.

There are certain rules which must be obeyed when writing formal specifications for processing by the MLS tools. These rules help the specifier express the security properties to be verified in the specifications. For example some of the rules for the MLS tool, as summarized by [SILV83], are:

- Declare a security level for every function reference.
- Define a binary partial ordering relation over the set of security levels.
- Express all flows of information in terms of the binary partial ordering relation.

In many cases the verification rules above introduce extraneous constructs into the specifications which can be counter-intuitive to a designer. Thus, one must be aware of the tradeoffs which are made when writing specifications for verification vs. design. An example of a problem of just this nature that arose in the process of writing the specifications for the SCOMP is as follows: the underlying security model on which the specifications were based assumed that all multilevel security checks were performed using a partial ordering relation. However, the implementation of the SCOMP system includes the concept of privilege, thus additional checks outside of the partial ordering relation had to be performed. The resulting specification then had a single multilevel security check for purposes of verification and a set of security and privilege checks for the system design.

One further consideration had to be taken into account when writing the specifications for verification. The verification system has bounds on both the space and time required to process a specification. Therefore, the specifiers had to be careful to include only enough complexity in the specifications so as to make them meaningful and yet manageable by the verification system.

The basic approach employed in the development of the specifications was to specify only the user-visible properties, that is, the functions at the user interface. This was consistent with the verification philosophy of the MLS tool, and the HDM approach to writing specifications for design verification. It should be noted though that in actuality the user-visible properties which were specified were the user-invokable functions. This is because there are certain user visible properties for which it is not possible to write SPECIAL specifications. In particular timing delays as a result of multiprogramming are visible to the user but are not formally specified. Thus in practice specifications are written in terms of the user-invokable functions rather than all truly user-visible properties.

The SCOMP kernel specifications include the effects of calls made by a user or trusted process to the 38 kernel gates and 12 hardware functions. These are specified in terms of all possible exceptions or returned error messages that could occur

and the effect if no error results. Many of the effects were specified by references to the effects of other lower-level functions. This nesting of effects often made the specifications difficult to verify since large numbers of formulas were generated.

Another problem that the specifiers encountered was how to include the concepts of privilege and tranquility in the specifications, since the MLS tools used in the verification did not include either concept. Privilege plays an important role in the implementation of the kernel. The protection mechanism of the kernel has special attributes used to denote privilege. These allow the simple security and star properties to be violated in certain instances, and are necessary for the kernel to function in an application environment. Examples of processes which possess privilege are operator, administrator, and auditor. In addition a real world application must have controlled upgrading and downgrading ability. Since the MLS tools do not incorporate privilege in the security checks, the verification tools reported security violations where in actuality a privilege check occurred. Consequently, whenever this happened comments were added to the specifications. The specifications then had to be manually checked against the failed formulas to insure that there were no additional security violations, or rather that the violations occurred in a controlled manner.

The inability to specify tranquility violations in the formal specifications was due to the MLS tool's assumption that all objects remain at a constant security level throughout their life in the system. Many applications need the ability to upgrade and downgrade an object's security level, thereby violating tranquility. The SCOMP kernel has two functions, "Set Device Access", and "Set Segment Access" which are designed to violate the tranquility principle, given that the caller has the proper privilege. Again comments were included in the specifications to this effect and manual checking had to be done.

In addition to insuring that the specifications adequately expressed the functionality of the kernel and its security requirements both from the viewpoint of the MLS properties and those other than MLS (tranquility and privilege), the evaluators analyzed the specifications in terms of completeness. In order for the verification results to be meaningful it is necessary that the specifications include all user-invokable properties of the kernel. If any user-invokable properties are not specified, then the verification and covert channel analysis will be incomplete.

IV. The Verification Effort

The kernel was verified using the HDM tools on three occasions. The first verification occurred in 1980 and the report on this effort is contained in [SILV81]. Then in 1982 a design change was made which required re-verification. This work was performed at The MITRE Corporation and is reported on in [EPST82]. However, neither of these first verifications was complete. In both attempts two kernel functions, "Create Process", and "Invoke Process" could not be processed by the MLS tool. The system on which the tools ran exhausted its address space before the complete set of formulas for the functions could be generated.

When the formal evaluation of the SCOMP system began in late 1982, it became evident that the remaining two functions would have to be verified in order for the SCOMP system to meet the A1 requirements of the Trusted Computing System Evaluation Criteria for design specification and verification. Honeywell requested assistance from the research group of the DoDCSC to complete the verification. The difficulty

in verifying the functions had been due to the fact that they were very long and complex, and had many nested "EFFECTS OF" statements. In order for the tools to process these functions, they had to be split into several subfunctions. Bret Hartman and Grant Wagner of DoDCSC developed a method to complete the MLS processing of these functions. Their method was shown to yield equivalent security information flows. A report documenting this effort is in progress. "Create Process", and "Invoke Process" were each broken up into eleven subfunctions, each of which contained all of the "EXCEPTIONS", but only one of the "EFFECTS OF" statements. All of the subfunctions were successfully processed by the MLS tool. In addition, all of the previously verified kernel functions were re-verified in order to re-create the complete proof files. A report on this third verification effort is in progress.

V. Analysis of The Verification Results

For each of the verifications performed, a report was made in which the verifiers analyzed the failed formulas to determine if an actual information flow existed. The evaluators were then presented with the formal top level specification (FTLS), terminal session transcripts of the proofs, and the analysis of the failed formulas. The evaluators then performed several tasks. First they checked that all specified functions had been assigned proper access levels. This is a critical step, since if access levels are improperly assigned the MLS formula generation could produce invalid formulas. Then they checked that the correct parameter had been selected as the access level for each function.

The MLS tool produces formulas based on its analysis of the specifications. In order for the specifications to be proven consistent with the MLS properties, all of the formulas must be proven by the theorem prover. In general, a failed formula indicates that the specification may violate the MLS properties. However, there are instances where a failed formula does not indicate an information flow, or when such an information flow may be acceptable.

In the case of the SCOMP system, all three of the verifications produced failed formulas. Honeywell provided English justifications for all of them, and the evaluators analyzed these justifications using a three-part process, as follows:

1) Each failed formula from the proof transcript was matched to Honeywell's English justification, thereby insuring that all failed formulas were justified.

2) The cause of the failed formulas was traced back to the kernel specifications, which gave the team a mapping between all of the Honeywell justifications and the information flows in the specifications.

   It should be noted however that the MLS tool did not always provide sufficient information to easily trace the failed formulas. In these cases the formulas had to be generated by hand in order to discover which information flows caused the failed formulas.

3) It was determined if each justification was valid. Valid justifications fell into two categories:

   a. Those formulas that failed because the theorem prover did not have enough information, or was unable to construct the proof, in which case the team checked that a proof could be constructed manually given the additional information contained in the justification.

b. Those formulas that failed because they were false, thereby revealing an information flow. In this case the team verified that the justifications had adequate explanations for the flow (e.g., closed by privilege checks) or that the flow channel was closed or its bandwidth minimized. This frequently involved checking the specifications against the code to insure that the bandwidth limitations specified in the justifications were properly implemented.

It is interesting to examine the statistics for the overall kernel verification. Fifty kernel functions (hardware and software) were formally specified and verified. The specifications for these functions consisted of approximately 3300 lines of SPECIAL which corresponded to 10,000 lines of PASCAL. The initial verification produced 2002 formulas. The formula generator eliminated 192 duplicate formulas and 1743 trivially true formulas. Of the 2002 formulas generated, only 67 were processed by the theorem prover, 34 of these were proven true and 33 were not provable. The second verification produced slightly more formulas but the number of unprovable formulas was identical. The third verification produced a similar number of formulas to the first two for the previously verified modules, and an additional 3224 formulas for the two previously unverified modules. Of the 3224 new formulas, 2970 formulas were eliminated as either trivially true or duplicates, 219 were proven true and 35 were unprovable. Thus, in the final verification 5226 formulas were generated, 4905 were eliminated, 253 were proven true and 68 were unprovable. The flows which caused the 68 failed formulas will be discussed under "Analysis of Covert Channels".

The number of formulas generated can at first be misleading, particularly in the case of the two previously unverified functions. These two functions generated a large number of duplicate formulas due to the redundancy that occurred when the functions were split into subfunctions. In addition one design flaw or covert channel can result in numerous failed formulas. In one case a single flaw caused 28 failed formulas. The channel could be exploited in 4 different ways, yet it appeared in 28 forms in the system. Thus, the total number of false formulas in a verification should not be considered a measure of the security of the system.

VI. Specification-To-Code Correspondence

The Trusted Computer System Evaluation Criteria requires manual mappings of the formal top level system specifications (FTLS) to the implementation source code for class A1 systems. One approach to specification-to-code mappings consists of mapping the English language descriptions in the Type B5 specification to the SPECIAL functions in the FTLS which then are mapped to pseudo-code descriptions in the type C5 specification and finally to the source code. Although several informal methods have been developed, notably, [SOL082], there currently exists no formal methodology for establishing these mappings. Given that this was the first formal evaluation of an A1 system and given the lack of proven methods for performing these mappings, the members of the formal verification evaluation subteam worked closely with Honeywell in developing methods and evaluating the correspondences. Initially, the team supplied Honeywell with interpretations of the Trusted Computer System Evaluation Criteria. This early effort resulted in the production of [BONN82]. This document mapped paragraphs in the Type B5 spec to functions in the FTLS, which were then mapped to paragraphs in the Type C5 specifications and functions in the source code modules. The formal verification evaluation team assisted Honeywell in further

refining these mappings. A procedure was developed which correlated data structures in the formal top level specification language, SPECIAL, to the data structures in the definition and types modules of the source code. Then each specified SPECIAL exception was mapped to each error condition in the source code. Finally the "EFFECTS_OF" statements in the SPECIAL specification had to be mapped to one or more statements in the code which actually implemented the effects of the function.

The mappings established are top-down, that is, all statements in the formal top level specification are mapped to lines in the implementation, but not all lines in the implementation are mapped up to the formal top level specification. This is because many implementation specific details are introduced in the lower level specifications and implementation code. However, it is important to ascertain that the implementation details do not map up and furthermore that all user-visible security relevant code is specified.

There are several important factors to consider in order to make such a mapping feasible. First, the naming conventions used in the formal top level specification must be consistent with those used in the implementation. Secondly, this procedure is very dependent on the semantics of the specification language used. Lastly, the use of comments in the implementation language is needed to simplify the task.

The result of the specification-to-code correspondence is an annotated top level specification. Seven discrepancies between the code and the FTLS were found as a result of this mapping. The evaluators met with Honeywell to analyze these, and all but two could be fully justified by Honeywell. The two unjustified discrepancies were due to exceptions, or error conditions, in the implementation code that did not appear in the FTLS, and both are known covert channels not found by the MLS tools. If these exceptions had been included in the FTLS, then the MLS tools would have found the corresponding channels. It should be noted that if these channels had not been identified previously by informal methods, then the specification-to-code correspondence would have found them. Thus, the process of establishing a correspondence between the specifications and the implementation code can assure that the implementation was derived from the verified specification, assist in locating covert channels, and assist in tracing formulas for the kernel verification analysis.

The process of establishing specification-to-code correspondence is tedious and time consuming. Because in this case four separate documents (B5-Spec, FTLS, C5-Spec, Source Code) were consulted in order to establish the correspondence, four people were used. Each person was responsible for two levels of specification (e.g., B-Spec and FTLS), to trace verbally the mapping of a particular structure through the levels.

The specification-to-code correspondence procedure for the SCOMP kernel took approximately 320 man hours to complete. This was able to be minimized thanks to several factors: the level of abstraction of the FTLS and the implementation was similar; the implementation was in Pascal which is a highly structured and typed language; and all members of the team were familiar with the specification and implementation languages.

VII. Analysis of Covert Channels

The analysis of covert channels is dependent on the analysis of the kernel verification and on the specification-to-code correlation. First, the kernel verification must detect the covert channels; then the specification-to-code correlation must insure that the channels have been closed or their bandwidths minimized in the implementation.

Perhaps one of the most interesting statistics to examine in this verification effort is the number of covert channels that were discovered through the use of the MLS tools as compared to those discovered through more informal manual methods. Currently, there are 12 known covert channels in the SCOMP system; other covert channels were discovered but closed. The bandwidths of the 12 remaining covert channels have been tested, all are fully audited, and their bandwidths have been minimized. Of these twelve, only four covert channels and one timing channel were discovered through the use of the MLS tools.

COVERT CHANNELS FOUND BY FORMAL METHODS

The existence of 68 unproven formulas was fully justified by Honeywell and checked by the evaluators. In theory, each failed formula denotes an invalid information flow in the system. In reality only 14 of the failed formulas denoted covert channels. These 14 covert channels were the result of resource exhaustion. One of the failed formulas occurred 11 times due to the splitting of "Create Process" into the subfunctions, and thus in actuality there were only 4 covert channels found by formal analysis. Measures have been taken to reduce the effective bandwidths of these channels in the implementation. The channels, their cause, and their bandwidths are described in a Honeywell technical note. Sixteen of the failed formulas (actually the same one in 16 places) were the result of a timing channel. Although the MLS flow analysis tools do not normally detect timing channels, this one was detected because the specifications included the relevant state variable information which is modified by the caller of the kernel gate. The timing channel is a result of the SCOMP system's "Exclusive Use of Segments" feature, which allows a user to obtain exclusive use of a segment. However, if another user already has exclusive use of the segment, then the user requesting exclusive use will block until the segment is released. The bandwidth of this channel is less than the maximum bandwidth for covert channels suggested by the Trusted Computer System Evaluation Criteria. The system designers feel that there is no means of eliminating the channel and that the "Exclusive Use" feature is critical for some applications.

The existence of the remaining 38 failed formulas was justified by Honeywell as explained below. Two of the failed formulas were a result of tranquility-principle violations. That is, the security level of an object is changed by a call to a kernel gate. These channels were closed by checking that the calling process had the necessary privileges. Twenty-seven of the failed formulas were a result of upgraded objects; these channels were closed by performing an additional "Read Check" and only returning information when the calling process had the proper access. Nine of the formulas could not be proven by the theorem prover because it did not have sufficient information. All of these were proven by hand using the additional information in the justifications. It is ironic to note that one of these failed formulas resulted because the theorem prover had oversimplified the formula to a point where it could not prove it.

The analysis of these formally found covert channels consisted of tracing the failed formulas back to the specifications, and then tracing the lines of the specification back to the lines of code. It is important to note that if the covert channel analysis had been performed prior to the specification-to-code correspondence, this task would have been much more difficult. Once the implementing lines of Pascal code had been located, the evaluators checked that the proper bandwidth minimizing techniques had been implemented and that the channel was audited. It must be stressed that in order for the kernel verification analysis, the covert channel analysis, and the specification-to-code correlation to proceed the specification-to-code correlation must be established first, then the kernel verification analysis, and finally the covert channel analysis.

COVERT CHANNELS FOUND BY INFORMAL METHODS

In addition to the 4 covert channels and the 1 timing channel found through the use of the MLS tools, 7 other covert channels have been identified. These channels were discovered during the process of design review and code walk-throughs. The channels are very similar to those found by the MLS tool, in that they are principally resource exhaustion channels. These channels were not found by the MLS analysis because of the manner in which the filesystem resources were modeled in the specifications. A filesystem was modeled in the specifications as capable of supporting only a fixed number of segments. Thus, the specification model of a filesystem did not reflect the fact that fixed resources are required to define a segment and variable resources are required to support the segment contents. Therefore, the resource exhaustion channel which occurs as a result of creating a segment was detected by the MLS tool. However, the resource exhaustion channel which occurs as the result of extending an existing segment was not detected. Honeywell is modifying the specifications so that they will better model the filesystem. Two other channels were not detected by the tools, because the specifications did not include a limit on the number of devices which the system can support, and the dependency between successive values of unique identifiers. The specifications are being modified to include both of these and will be re-verified. It is believed that these channels will be detected by the MLS tools when the modified specifications are verified.

The evaluators' method of analyzing these informally found channels focused on the code rather than on the specifications. Furthermore, the implementation code was closely scrutinized for the existence of additional covert channels. The ratio of formally found covert channels to informally found covert channels (5 to 7) is somewhat disturbing. One might naturally ask how were these channels found and how do we know that indeed all were found? Unfortunately, we have no assurance outside of penetration testing that all covert channels were found. However, if the specifications were modified as above, then it is believed that all of the identified covert channels would have been located by the MLS tools.

Several other points should be made. First, we can be reasonably sure that all covert channels have been identified largely due to the experience and expertise of the system developers and design review board. Second, as we learn more about specification writing and benefit from the experience of other verification efforts, this knowledge will contribute to specifications better tailored to detecting covert channels. Clearly, guidelines on specification writing and covert channel detection are needed.

VIII. Conclusions

The SCOMP system was the first security kernel of its size and complexity to be formally verified, and the first to be evaluated against the Trusted Computer System Evaluation Criteria. The knowledge gained from this effort will contribute to future verification and evaluation projects.

This paper has examined some of the types of problems encountered in the SCOMP system verification. The methods and procedures developed for analyzing the verification, notably in the area of covert channel analysis and specification-to-code correspondence, have been presented. A statistical analysis of the kernel verification results and covert channel analysis was also presented.

In the end, it was felt that the verification successfully provided additional assurance of the security of the system. The specification-to-code correlation was very helpful in convincing the evaluators that the verified system was the implemented system, and providing guidance to the penetration team. Specification-to-code also contributed to the analysis of the verification results and to the covert channel analysis. Four covert channels and 1 timing channel were detected by the MLS tools that may not have been found through informal analysis. In most cases, an informal analysis would have found fewer covert channels. (However in this case the system developers and the initial design reviewers were very experienced and knowledgeable in multilevel security and kernel design.) Although the ratio of formally found to informally found covert channels was low, it is believed that more channels could have been found formally. Thus it has been demonstrated that the use of formal methods is significant in analyzing a system's security.

Acknowledgements

The kernel design verification discussed in this paper was performed by the Honeywell Corporation, principally John Silverman and Chuck Bonneau. The kernel was re-verified by Harvey Epstein of The MITRE Corporation. The evaluation and analysis of this work was performed by the formal subteam of the SCOMP evaluation team: Terry Vickers Benzel and Dave Drake of the MITRE Corporation, and Bret Hartman and Tad Taylor of the DoDCSC. This paper would not be possible without the efforts and technical contributions of these individuals.

REFERENCES

BELL74 Bell, D. E., LaPadula, , "Secure Computer Systems Mathematical Foundations and Model", M74-244, The Mitre Corporation, Bedford, MA, October 1974.

BENZ83 Benzel Vickers, T., "Overview of the SCOMP Architecture and Security Mechanisms", MTR-9071, The Mitre Corporation, Bedford, MA, September 1983.

BONN82 Bonneau, C. H., "SCOMP Specification-To-Code Correlation", Honeywell Information Systems, McLean, VA, 1983.

CSC83 "Department of Defense Trusted Computer System Evaluation Criteria", CSC-STD-001-83, Department of Defense Computer Security Center, Fort George G. Meade, Maryland.

DOD5200 DoD Directive 5200.28, Security Requirements For Automatic Data Processing (ADP) Systems, revised April 1978.

EPST82 Epstein, H. I., "SCOMP Kernel Re-verification Results", MTR-8781, The Mitre Corporation, Bedford, MA, September 1982.

FEIER80 Feiertag, R. J., "A Technique for Proving Specifications are Multilevel Secure", CSL-109, SRI International, Menlo Park, CA, January 1980.

FRAIM83 Fraim, , "SCOMP: A Solution to the Multilevel Security Problem", Computer, Volume 16, Number 7, July 1983.

ROBI79 Robinson, CA, June 1979.

SILV81 Silverman, J., "Proving an Operating System Kernel Secure", Technical Report 81SRC31, Honeywell Systems & Research Center, Minneapolis, MN, April 1981.

SILV83 Silverman, J., "Reflections on the Verification of the Security of an Operating System Kernel", Proceedings of the Ninth ACM Symposium on Operating Systems Principles, October 1983.

SOLO82 Solomon, J. "Specification-To-Code Correlation", Proceedings of the 1982 Symposium on Security and Privacy, Oakland California, April 1982.

THE AUTOMATED RISK PROFILE
(RiskPac™)

Peter S. Browne
Profile Analysis Corporation

An ADP risk assessment looks at an organization's ability to perform operational functions in a correct and timely manner. It involves the review of a given ADP environment, an analysis of loss exposure and the development of minimum and recommended control standards. It should provide the criteria to determine an appropriate mix of access control, audit trails, transaction authentication and network security.

Presently, the technology of risk assessment is not adequate for modeling uncertain threats, uncertain exposures and uncertain implementation of safeguards. Even though the mathematical underpinnings are based on a solid actuarial science, the risk analytic process tends to obscure the results. All too often the veracity of a given threat frequency or loss exposure is cast in severe doubt. The input is subjective, the calculation and relationship process is complex and the output is usually not statistically valid.

The other major problem is that of cost/benefit. In order to produce valid results, a quantified risk assessment must rely on large masses of data. The collection is labor intensive, thus costly. When results do not match expectations, the whole method or process loses credibility. Thus, risk assessment is not the panacea once thought.

RiskPac™ was designed for the purpose of providing explicit guidance for decentralized users of distributed processing systems. In developing a "risk profile", this system facilitates the collection of important information about processing locations and applications software. The purpose is to determine an appropriate mix of security, audit and management control.

The system objectives are to:

- Assess impact of new ADP systems on an organization's ability to perform various business functions.
- Determine how those systems might fail to meet their functional objectives.
- Assess inherent risks due to the system environment and function.
- Provide guidance for implementation of appropriate controls.

Approach

The determination of risk is accomplished by focusing on computer applications systems. Questions are asked about connectivity, type of system, degree of access control, system criticality and sensitivity.

Various rating factors or "exposure quotients" are then assigned, based on answers to the questions. Depending on the calculated risk profile ratings, certain data security standards are then required or suggested.

The risk profile is a broad based management tool that can be used to initially determine risk exposures in a computer application, a physical location, or a business function. It will identify and describe risks, but will not calculate dollar loss exposure or threat probabilities. It also will not consider the completeness or consistency of controls in place.

The risk profile system does address issues of management concern. For example, is a new transaction processing system vulnerable to tampering? Does new technology such as networking require new controls? Is there an overall need to provide an awareness of risk and to force the decision to accept or reject it.

The following uses of the risk profile system are relevant:

- Review of a DP processing center for lack of security controls
- Analysis of an application system for sensitivity
- Decision as to whether to attach to a corporate data base
- Security evaluation of a networking implementation
- Evaluation of risk due to changes in a business function

The major purpose is to spread security awareness by allowing the software and procedures to be used by a multitude of user and data processing functional personnel, throughout the organization.

System Environment

The software resides on an IBM PC. Thus, it should be available in most organizational locations. Because its use will involve the assessment of data processing risk, input will be interactive. However, for those locations that do not have a PC, the system will be capable of accepting input from forms.

The primary user of the risk profile would be the ADP security officer of the system under study. However, it is also designed to be used by various system users, especially if the scope is to be focused on applications or operational functions.

Software Functions

The software provides five basic functions. They are to:

- Identify and collect data on the operating environment, by posing a series of on-line scoping questions.

  The questions are scored, and the user guided into providing more information to properly describe the scope of the problem, the major assets or function under analysis, and particular items of concern.

- Develop profiles of applications and their data, so that the criticality of the application and the sensitivity of the data or application to disclosure or manipulation can be determined. The process will be similar to current data and processing classification guidelines.

- Relate the physical, processing and applications profiles in terms of exposures, developing an "exposure quotient" rating for each asset under consideration.

- Assess risk in descriptive term, presenting tables, graphs and charts of inherent exposure.

- Develop lists of appropriate, tailored standards or control guidelines to follow in terms of assessing whether appropriate controls are in place for the given application or site.

Approach

The software is menu-driven. There is one user menu which provides access to the questionnaires, calculation, reporting, system installation and delete functions.

The heart of the system is a questionnaire, which can be tailored to each organization. Branching logic allows different paths to be taken based on yes/no questions, or specific answers of a multiple-choice question. Results are stored in a file for later retrieval, risk computation and mapping of control standards.

Questions are presented in a three level hierarchy.

- The processing environment level asks questions about the type of DP system and its environment.

- The processor level asks questions about the nature of the individual hardware, physical environment, communcations and access controls.

- The applications level asks questions about the individual applications systems, their inherent risk and the nature of access controls.

The survey may be completed in increments. There is the capability to stop answering the questions, store the intermediate results in a file, and then continue where left off at another session. A maintenance capability allos the changing of answers for purposes of correction or for modeling changes in the environment, the system, or controls.

Question responses are edited for logical consistency, to include required response, alphabetic or numeric answer, field size, etc. A series of seven reports are produced. Each deals with the risk factors of the environment, processor or application.

- R-1 - Describes the processing environment.

- R-2 - Describes the physical profile of a given processing machine.

- R-3 - Shows all applications under analysis and keys them to their processor.

- R-4 - Describes the overall risk of the given application.

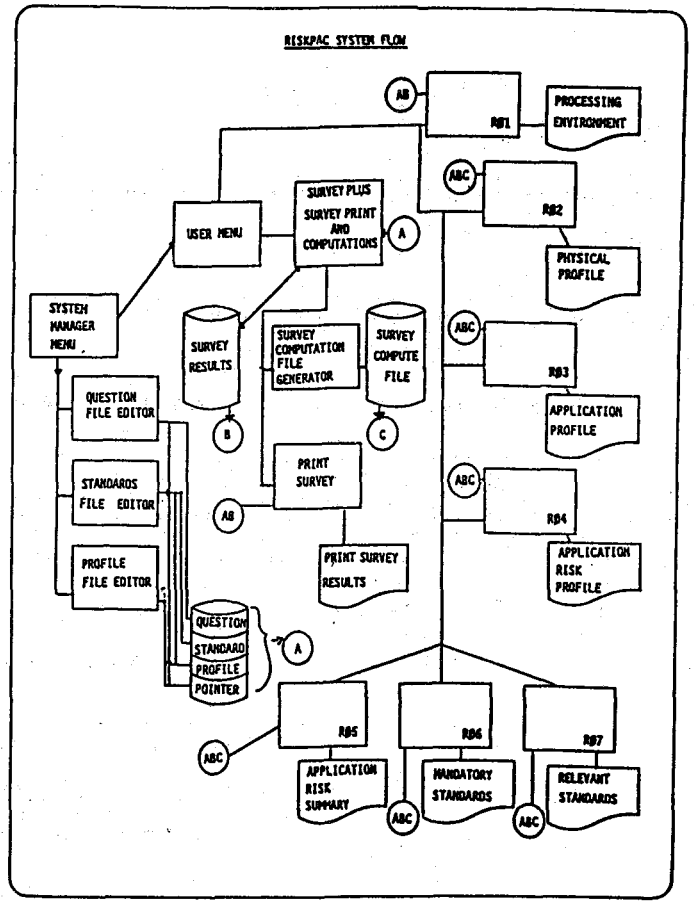
- R-5 - Summarizes risk across all applications under analysis.

- R-6 - Prints mandatory requirements for control.

- R-7 - Maps the risk level to data security standards and guidelines. This report can be produced at the application or the individual processor level.

The reports are designed to be printed on any standard printer.

The following pages show the overall system flow, and provide a sampling of some of the user input screens.




```
                    ***  RiskPAC  ***
                      SYSTEM  INSTALLATION


     What DISKETTE drive will be used for SOFTWARE      (a-c):  c
     What DISKETTE drive will be used for SYSTEM FILES (a-c):  c
     What DISKETTE drive will be used for DATA STORAGE (a-c):  c
     ENTER Installation Title:  WORKINGDEMO
     Enter TITLE OPTION (1 - double size, 2 - standard size):  1
```

```
                    ***  RiskPAC  ***
                      REPORT  MENU


          F1  - Run Report 01 - PROCESSING ENVIRONMENT
          F2  - Run Report 02 - PHYSICAL PROFILE
          F3  - Run Report 03 - APPLICATION PROFILE
          F4  - Run Report 04 - APPLICATION RISK PROFILE & CALCULATIONS
          F5  - Run Report 05 - APPLICATION RISK SUMMARY
          F6  - Run Report 06 - MANDATORY STANDARDS

          F8  - Run Report 08 - RELEVANT STANDARDS

          F10 - EXIT from REPORT MENU

     Please Select a FUNCTION Key
```

```
                    ***  RiskPAC  ***
                        USER MENU
     Enter SURVEY ID:  001                  CREATION DATE:  09-12-84
     Enter SURVEY NAME:  DEMONSTRATION      RiskPAC Disk:  c
     Enter Your NAME:  P. S. Browne         Data Disk:  c


          F1  - SURVEY Questionaire

          F3  - REPORT Menu
          F4  - SPECIFY A SURVEY NUMBER


          F8  - SURVEY Deletion

          F10 - EXIT from Risk Profile System
     SURVEY EXISTS
     Please Select a FUNCTION Key
```

A BAYESIAN APPROACH TO THE ASSESSMENT OF RISK FOR COMPUTER AND COMMUNICATION SYSTEMS *

Ali Mosleh
Pickard, Lowe and Garrick, Inc.
2260 University Drive
Newport Beach, California 92660

INTRODUCTION

Concern for the security and reliability of information processing and communication systems has been rapidly rising in recent years. The reasons are obvious. An increasing number of organizations are becoming almost totally dependent on computers and communication technology without being adequately prepared for the potential risks involved. These systems are vulnerable to threats, initiated accidentally or deliberately, with consequences beyond acceptable limits. Managing the new generation of risk to the modern organizations, therefore, has become an important and challenging task for management.

The first and most important step in effective risk management is to have an assessment of the risks involved. Risk assessment of any automated environment requires understanding and analysis of the interrelationships among the system's primary elements as well as the vulnerabilities of each of these elements and the threats that could affect them. Vulnerabilities affecting computers and communication systems generally fall in the following categories: physical surroundings, hardware, system software, communications links, and organizational personnel and procedures. Threats to the system can impact one or more of these areas of vulnerability (also called "exposures"). A risk analysis is performed to identify the threats and vulnerabilities of the system, assess the likelihood and the magnitude of their impact, and provide necessary input for decision-making with regard to alternative actions in dealing with such threats and vulnerabilities.

Risk assessment methodologies currently in use fall into two general categories: qualitative and probabilistic. Qualitative methods (Reference 1) consist of identifying all the threats to the system and then establishing to which of the system's resources they apply. Although threats and vulnerabilities may be compared using qualitative methodologies, no measure is provided to evaluate the threats according to their seriousness, and there is no indication of how cost effective it will be to correct various vulnerabilities. Although qualitative methods are useful in identifying threats and vulnerabilities, little value can be attached to them as aids to meaningful decision-making and risk management. It is worth mentioning that there are other "qualitative" techniques which, in addition to identifying the threats and vulnerabilities, provide a subjective ranking of their risk significance. Although such techniques are of more practical value to a decision-maker they do not provide the necessary information about the magnitude of risk for cost-benefit analysis and evaluation of alternative courses of action.

The probabilistic methods, on the other hand, provide a full measure of the seriousness of each threat and the magnitude of its impact. This is done by recognizing that risk is composed of two basic elements: (1) a damage or loss, and (2) the uncertainty about whether the damage or loss will be received. Each of these elements is quantified based on the best available evidence for each threat and vulnerability.

The critical element in determining the validity, appropriateness, and accuracy of a quantitative risk analysis is the manner in which the uncertainties are accounted for. Unfortunately, most probabilistic methodologies used today employ point estimates; i.e., one specific number is used to represent the frequency of an undesirable event rather than calculating a probability range for the varying degrees of certainty in the estimated frequency (References 2 and 3). Any methodology that relies solely on point estimates of quantitative data is too sensitive to possible variations in the basic input to risk models. Errors in calculating the level of risk occur because a fixed frequency

or magnitude of frequency (high, low, moderate) is assumed for the rate of occurrence of the various threats to the system, and because the formulation of loss estimates requires that the cost of recovery be a specific dollar figure. Unfortunately, threat frequencies can vary considerably and it is often impossible to predict recovery costs precisely beforehand.

In contrast, the probabilistic risk assessment methodology of this paper is based on Bayesian statistics and can deal with a range of threat frequencies as well as a range of safeguards or recovery costs, and its mathematical formulation is capable of taking into account multiple threats against a given vulnerability over a given period of time in the system. For selected levels of computer security, this methodology provides a decision-making tool by displaying the range of cost-benefit options as readily understandable "risk curves" and/or in tabular format. The quantification of the variation in the cost, benefits, and risks associated with each option provides a much stronger basis for decision-making.

BAYESIAN PROBABILISTIC RISK ASSESSMENT

Bayesian risk assessment methodologies were originally developed for use in the nuclear power industry where consequences can be devastating and risk analysis models must be of unquestioned mathematical validity (Reference 4). This methodology can also be used as a systematic, quantitative approach to the evaluation of financial risks. The quantitative understanding of risk is obtained by providing answers to four fundamental questions:

1. What can go wrong?
2. How frequently can it be expected to happen?
3. What would be its consequences?
4. How certain are we about the answers to the first three questions?

The computer system of an organization is subject to many threats. To answer the first question, a number of potential threats can be enumerated, such as fraud, malicious vandalism, fires, earthquakes, system unavailability due to component failures, etc. Although each of these threats may or may not materialize in a given organization, historical evidence and industry experience indicate that each one is real and can be expected to happen to any computer facility. This evidence can be used to provide estimates for frequency of the threats and the magnitude of the losses due to any of these threats (Reference 5).

For a number of reasons, such as lack of sufficiently extensive historical data or variations of the parameters of the problem with time, it is not always possible to provide totally accurate estimates of threat frequencies and damage levels. (In fact, there is always some degree of uncertainty regarding the real magnitude of these quantities.) As pointed out before, although many methodologies currently available for risk assessment ignore the issue of uncertainty in various steps of a risk analysis, it is clear that uncertainty is a component of risk and is essential to its conceptually correct quantification. Therefore, question 4, concerning the degree of certainty, must also be dealt with. To do so, a Bayesian analysis is made of the available statistics to develop a mathematically appropriate "spread" for estimating the frequency and the consequences of a given undesirable event. The mathematical model is described in the following.

MATHEMATICAL MODEL

We will assume that there are N potential threats to the system and each threat occurs with a specific frequency. According to our discussion in the previous section, the exact frequencies of threats are not usually known. Let

$f(\lambda_i)$ be a distribution representing our uncertainty about the actual value of the frequency of occurrence of threat type i. Later, we will discuss how various types of information can be used to assess such distributions.

We further assume that the threat arrivals follow a Poisson process. That is, the probability of having k_i threats of type "i" during a time period of "t" is (Reference 6)

$$P_t(k_i|\lambda_i) = \frac{1}{k_i!}(\lambda_i t)^{k_i} e^{-\lambda_i t} \quad k_i = 0,1,\ldots\infty \tag{1}$$

If the value of λ_i is known exactly, the above equation gives the probability of k_i. The unconditional probability of having k_i threats of type "i" is then given by

$$P_t(k_i) = \int_0^\infty P(k_i|\lambda_i)f(\lambda_i)\, d\lambda_i \tag{2}$$

Suppose now that for each threat occurrence we have a potential loss of C_{ij} (for the jth occurrence of type i threat). The total cost is then

$$C_i = \sum_{j=1}^{k_i} C_{ij} \tag{3}$$

Again, we need to model our uncertainty about the magnitude of the loss due to occurrence of each threat. Let $g(C_{ij})$ represent the uncertainty on C_{ij}. The distribution of C_i, the total loss due to threat i, can then be obtained by convolution of g's based on Equation 3. Note that the resulting distribution [call that $(C_i|k_i)$] is conditional on the number of occurrences of type i threat. The unconditional probability of loss due to threat i can be calculated from

$$p_t(C_i) = \sum_{k_i=0}^{\infty} f(C_i|k_i)P_t(k_i) \tag{4}$$

This distribution cannot be found analytically and the computations should be performed by numerical methods. The cumulative distribution of C_i is given by

$$P_t(C_i) = \sum_{k_i=0}^{\infty} \phi(C_i|k_i)P_t(k_i) \tag{5}$$

where

$$\phi(C_i|k_i) = \int_0^{C_i} f(C_i'|k_i)dC_i' \tag{6}$$

The risk curve from type i threats is now

$$R_t(C_i) = \text{Prob. (Loss in time } t \geq C_i) = 1 - P_t(C_i)$$

Figure 1 shows a typical risk curve. From this curve one can read the chance of exceeding a certain level of loss in a specified time period in the future.

One may also find the total risk from all types of threats by calculating the distribution of the following random variable

$$C = \sum_{i=1}^{N} C_i \tag{8}$$

This can be obtained by convolution of the distribution of C_i's which we have already calculated. This type of calculation can be easily done by computers for different time periods and the final result would look like Figure 2.

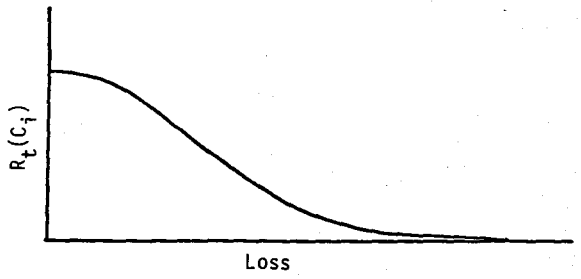


FIGURE 1. A TYPICAL RISK CURVE

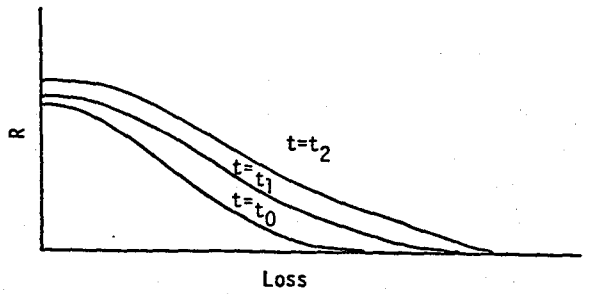


FIGURE 2. FINAL RISK CURVES FOR DIFFERENT TIME PERIODS

To compare the risk from different threats, the decision maker can plot the corresponding risk curves on the same graph and examine them with his or her risk acceptance criteria (see Figure 3).

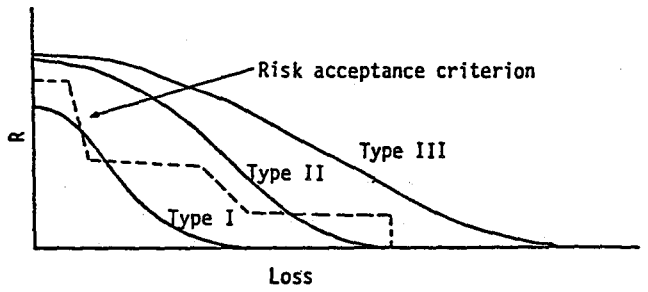


FIGURE 3. RISK COMPARISON FOR VARIOUS THREATS

In Figure 3, as we can see, threat type III is the most serious and totally unacceptable, whereas type I has a lower probability of causing high loss and is mostly acceptable according to the risk acceptance line.

To observe the effect of applying protective measures for each threat, a second risk curve for each threat type is needed which includes the initial cost of the protective measure as well as the changes in the total cost due to the

presence of that protective measure. A typical case is shown in Figure 4. The mathematics are basically the same as before and will not be repeated here.

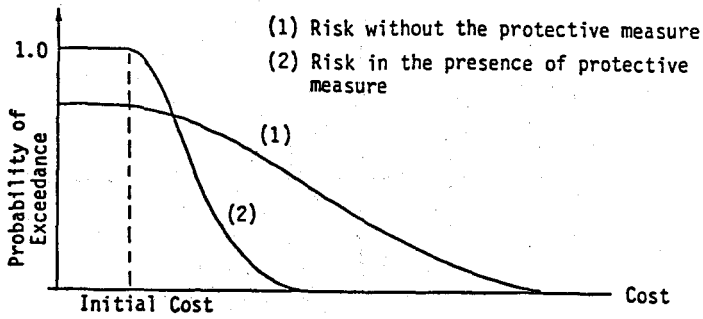


FIGURE 4. THE CHANGE IN RISK FOR THE NEXT t YEARS DUE TO APPLICATION OF PROTECTIVE MEASURES

ASSESSMENT OF THE PARAMETERS OF THE MODEL

The validity of the results of a risk assessment is a direct function of the accuracy of the input to the risk model. The input parameters in the modes of this paper are the threat frequencies and the value of loss for each threat. Depending on the type of environment being analyzed and the availability of information, estimates for such parameters may be difficult to obtain. For example, although the frequency of physical threats such as fires or floods may be assessed with a reasonable degree of accuracy, information on the likelihood of deliberate impairment of the system or malicious misuse of its resources is not easily obtainable. Similarly the assessment of the value of tangible assets is a much simpler task as compared with the evaluation of the value of information loss or denial of access.

Quantitative risk assessment methods currently in use have, unfortunately, failed to provide an adequate framework for dealing with situations where uncertainty plays a key role and where there is a need to be able to make use of both the statistical evidence and the opinion of experts.

The advantage of the Bayesian methods in this respect is that they provide powerful and systematic approaches to handling information about uncertain quantities for which little "hard" evidence is available.

The key issue in developing distributions for the parameters of the probabilistic risk assessment models is to guarantee that the information regarding each parameter, its relevance, and its value as viewed by the analyst are presented correctly and that various pieces of information are integrated coherently. "Coherence" is preserved if the final outcome of the process is consistent with every bit of information used and all assumptions made. This is done by utilizing the fundamental tool of probabilistic inference; i.e., Bayes' theorem (Reference 8). Mathematically, Bayes' theorem is written as

$$P(x|E,E_0) = k^{-1} L(E|x,E_0)P(x|E_0) \tag{9}$$

where

$P(x|E,E_0) \equiv$ probability of x being the true value of an unknown quantity in light of new evidence E and prior body of knowledge E_0.

$L(E|x,E_0) \equiv$ likelihood of the new evidence E assuming that the true value is x.

$P(x|E_0)$ ≡ probability of x being the true value of the unknown quantity based on the state of knowledge E_0 prior to receiving E.

Finally, k is a normalizing factor defined as

$$k \equiv \int_{\text{all } x} L(E|x,E_0)P(x|E_0)dx \tag{10}$$

Recent advancements in techniques for using expert opinion for assessing unknown quantities (Reference 9), combining statistical data with subjective information (Reference 10), and improving "generic" estimates by incorporating new system specific evidence (Reference 11), have proven to be useful in quantitative assessment of risk in other technologies. Such techniques can be, and to a limited extent have been, used in the data processing and communication networks, as discussed briefly in the following example application.

EXAMPLE APPLICATION

The methodology of this paper was used in the analysis of the communication security environment of a large computer network (Reference 7). The focal point of the analysis was the encryption keys used for communication. As a result of the investigations, several threats and vulnerabilities were identified and the magnitude of the potential consequences was quantified. For each threat, a distribution of frequency was developed based on available statistical evidence (Reference 5). Figure 5 is an example of how uncertainty regarding the frequency of one of the threats was represented by a probability distribution (Reference 8). Figure 6 presents the calculated risk for two identified threats in the form of risk curves that give the probability of various levels of loss in a 1-year period. It can be seen, for example, that there is a 75% chance that the annual loss due to threat 1 exceeds $1,000,000. At the same time, the chance of losing $1,000,000 due to threat 2 is essentially zero.

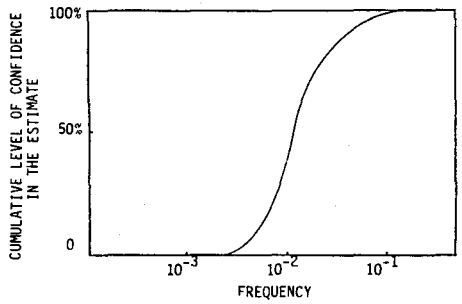


FIGURE 5. ANNUAL FREQUENCY OF BANK FRAUDS PER BANK EMPLOYEE

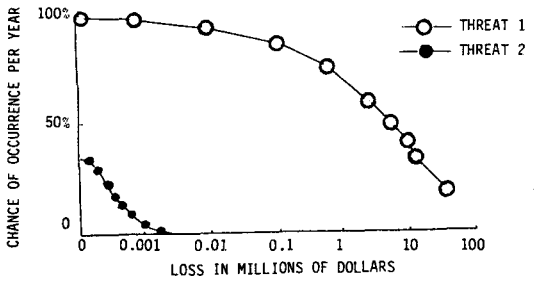


FIGURE 6. RISK CURVE FOR TWO DIFFERENT THREATS

REFERENCES

1. Brafman, M. J., "Evaluating Computer Controls Using the Matrix Approach," The EPD Audit, Control and Security Newsletter, December 1981.

2. Courtney, R. H., "Security Risk Assessment in Electrical Data Processing Systems," Proceedings of the National Computer Conference, 1977.

3. Lobel, J., "Risk Analysis in the 1980's," Proceedings of the National Computer Conference, 1980.

4. Kaplan, S., et al, "Methodology for Probabilistic Risk Assessment of Nuclear Power Plants," Pickard, Lowe and Garrick, Inc., PLG-0209, 1981.

5. Pickard, Lowe and Garrick, Inc., Proprietary Data, 1982.

6. Cinlar, E., Introduction to Stochastic Processes, Prentice-Hall, Englewood Cliffs, N.Y., 1975.

7. Winkler, R. L., and W. L. Hays, Statistics, Holt, Rinehard and Winston, N.Y., 1975.

8. Lindley, D. V., Introduction to Probability and Statistics. Part 1: Probability, Part 2: Inference, Cambridge University Press, 1970.

9. Mosleh, A., and G. Apostolakis, "Models for the Use of Expert Opinions," Proceedings, Workshop on Low-Probability/High-Consequence Risk Analysis, Arlington, Virginia, June 15-17, 1982, Plenum Press, New York, 1983.

10. Apostolakis, G., and A. Mosleh, "Expert Opinion and Statistical Evidence: An Application to Reactor Core Melt Frequency," Nuclear Science and Engineering, 70, 135-149, 1979.

11. Apostolakis, G., "Data Analysis in Risk Assessments," Nuclear Engineering and Design, 71, pp. 375-381, 1982.

12. Mosleh, A., E. R. Hilton, and A. F. Gersman, "ATM Key Management Risk Analysis," Pickard, Lowe and Garrick, Inc., PLG-0233, prepared for Target Bank, 1982.