M79-225

PROPOSED TECHNICAL EVALUATION CRITERIA FOR TRUSTED COMPUTER SYSTEMS

G. H. Nibaldi

25 October 1979

[illegible] ISIC

[illegible]

RFC

THE MITRE CORPORATION
Bedford, Massachusetts

Contract No: AF19628-80-C-0001
Contract Sponser: OUSDRE (C^3I)
Department: D-75

ABSTRACT

The DoD has established a Computer Security Initiative to foster the wide-spread availability of trusted computer systems. An essential element of the Initiative is the identification of criteria and guidelines for evaluating the internal protection mechanisms of computer systems. This report documents a proposed set of technical evaluation criteria. These criteria and any evaluation process that they might imply represent one approach to how trusted systems might be evaluated.

The information in this report is made available to stimulate technical discussion among industry and government personnel. Procedures or criteria formally coordinated and adopted by the Department of Defense in the future may differ from those proposed here. The views and conclusions contained in this paper are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Department of Defense or the United States Government.

ACKNOWLEDGMENTS

The author would like to thank the people who helped in finalizing the ideas presented in this paper: Jack Adams (IBM), Pete Tasker, Steve Walker, (OUSD C3I), and the participants of the Secure System Evaluation session of the 1979 Summer Study on Air Force Computer Security.

Special thanks go to George Miller for his editorial help and to Karen Borgeson for her secretarial support while learning a new text processing system.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

SECTION 1

INTRODUCTION

Trusted computer systems are operating systems capable of preventing users from accessing more information than that to which they are authorized. Such systems are in great demand as more processing is entrusted to computers while less information should be shared by all the system's users. With this demand comes a need to ascertain the integrity of computer systems on the market. As part of the Department of Defense Computer Security Initiative [1], a plan has been devised for this purpose. Under this plan, computer systems will undergo "laboratory evaluations," where their suitability for different types of operational environments can be analyzed. A proposed set of evaluation criteria to be used in such an analysis is documented in this report.

BACKGROUND

In multi-user systems, the underlying assumption has been that malicious users or their programs may attempt to access information to which they would not normally be entitled. Operating systems* can potentially confine users so that unauthorized access cannot occur. On the other hand, if incorrectly implemented, they have the potential to undermine any safeguards that might have been built into user programs or applications. By examining the strengths and weaknesses of a computer's operating system, one can draw conclusions about the suitability of the system for diverse environments (characterized, for instance, by degree of data sensitivity, criticality of functions, user community). Thus, the stronger the operating system, the less vulnerable the system to malicious attack.

A synopsis of the general computer security problem as well as the seminal work on evaluation criteria is reported by Lee et al. [1]. The reader is referred to that report for a historical perspective. The work that led to this report entailed fleshing out the details of an initial set of evaluation criteria presented in that document [1]. Specifically, the task was to:

*"Operating system" has also been referred to variously as executive, monitor, and supervisor. As used here, it includes the underlying hardware base in addition to software.

1. Identify the protection-related aspects of operating systems--not only the protection services but also the proof that the services are sufficient;

2. Determine their relative importance;

3. Establish thresholds that clearly distinguish the level of an operating system by the quality of its protection; and

4. Determine the environments that operating systems at each level could support.

OVERVIEW

This report will cover the first three points, by identifying the features of computer systems that contribute to internal protection,* and from them devising criteria for system evaluation. The fourth point is discussed by Lee et al. [1].

The protection-related features fall into three categories: policy, mechanism, and assurance. Policies provide the access rules under which the system is expected to operate. Mechanisms provide the foundation for policy enforcement. Assurances offer evidence that the mechanisms operate correctly.

The criteria are presented for seven hierarchical "levels of protection"--the intent is that the higher the level, the greater the system's protection. With the present state of technology, no one can claim absolute confidence in a computer's controls. Hardware limitations, the complexity of software, and the uncertainty of an environment all increase the likelihood of errors. The evaluation criteria described here attempt to address the known vulnerabilities of computer systems. These criteria are expected to grow and mature with our increasing understanding of computer protection.

*The protection of information in computer systems is commonly referred to as "data security."

SECTION 2

EVALUATION FACTORS

Many factors play a role in the perceived quality of internal protection of computer systems. Three general areas will be considered:

1. Protection _policy_,

2. _Mechanisms_ contributing to effective enforcement of the policy, and

3. _Assurances_ that the mechanisms are indeed functioning.

The various aspects of policy, mechanism, and assurance, and their relationship to each other, are depicted in figure 1.

PRIMARY FACTORS

Policy

Commitment to a protection policy is a prerequisite for a secure system. Without a clear statement of policy there is no way to determine if the system will meet even minimum requirements. In this section we review the basic elements of protection policy.

A protection policy outlines a set of guidelines for determining how computer resources in general and information in particular may be shared. The policy is presented in terms of well-defined rules that conform to some notion of "access"--by whom, to what, under what conditions, and how. Another way to define a protection policy is in terms of service: a protection policy prescribes the manner and conditions under which a subject (e.g., user, process) is served by the system. If we view the computer system as an abstract, high-level machine, the services are _operations_ to the system, equivalent to high-level machine instructions. The system determines, based on the policy, whether or not to perform the operation. It might allow a user to log in, execute a program, access an I/O device, or halt the machine. The conditions for performing the service may depend upon a characteristic (or state) of the subject or an object (e.g., files, tapes) involved in the service, upon the state of the system (e.g., number of users logged in), or upon some external factor (e.g., time of day).

IA56580

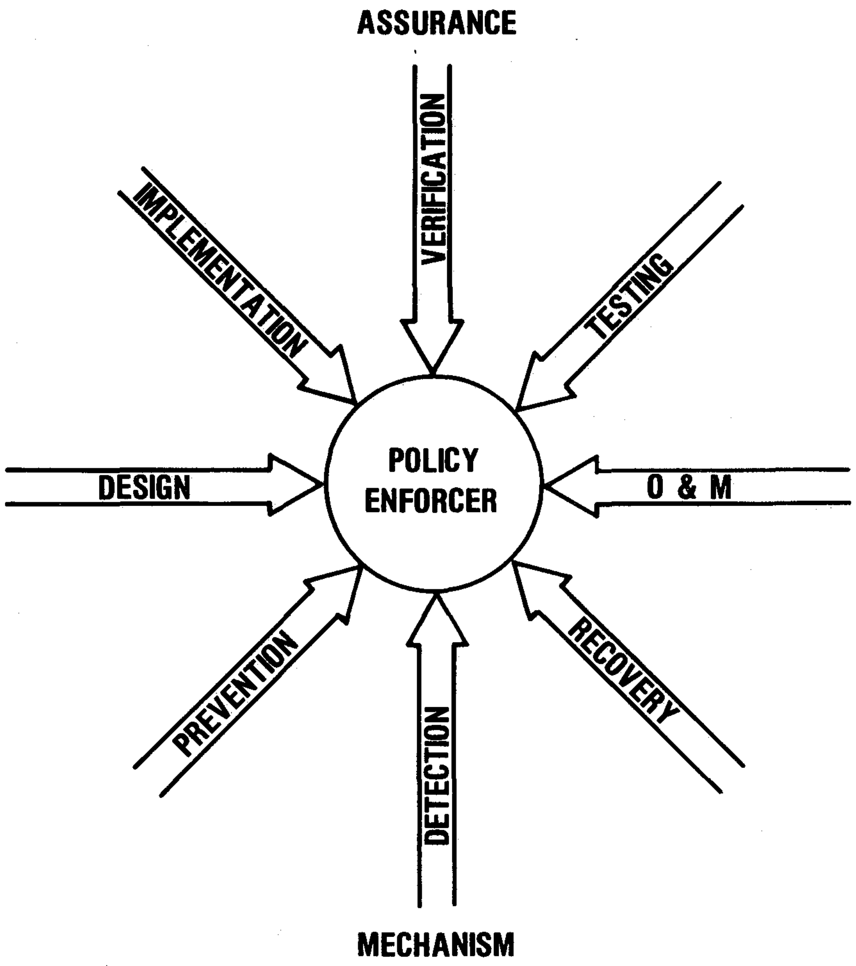


Figure 1. Factors of Trusted Operating Systems

If the service involves an information, or data, object (e.g., files, I/O devices) the policy will be referred to as a _data policy_. A data policy prescribes the manner and conditions under which a subject interacts with a data object. The manner of interaction defines operations on the objects. Examples are: read a file, access an I/O device, update a file, or change the owner of a file. The operations may concern either the contents of the object (as in read a file) or the state of the object (as in change the owner). Two distinct but significant aspects of data policy are recognized [2,3]: _access control_ and _flow control_. Access control relates to the manipulation of objects as containers of information (e.g., reading a file); flow control addresses how the contents of the objects may be passed from one object to another (e.g., copying a file). The conditions of a data policy specify either access control or the more restrictive flow control. The distinction between access and flow is depicted in figure 2.

If a service affects the "manner and conditions" of service, the policy will be called an _authorization policy_. An authorization policy prescribes the manner and conditions under which subjects may set authorizations for a given subject and object. For instance, there may be an authorization policy that allows the owner of an object to grant others access to that object.

An authorization policy may be characterized by degree of locality of control: a _mandatory_ policy is externally predetermined (e.g., by law); a _discretionary_ policy implies individual judgment (i.e., at the "discretion" of the user). A policy can be both mandatory and discretionary if authorizations may only be changed within preset limits.

Three specific policies that factor into an evaluation are:

1. A policy on information compromise (security policy);

2. The policy practiced within the Department of Defense and in the intelligence community (DoD policy); and

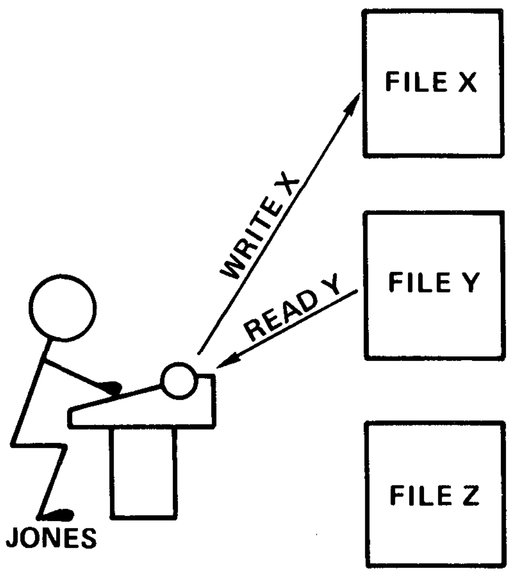
3. A policy regarding denial of service conditions.

Security Policy

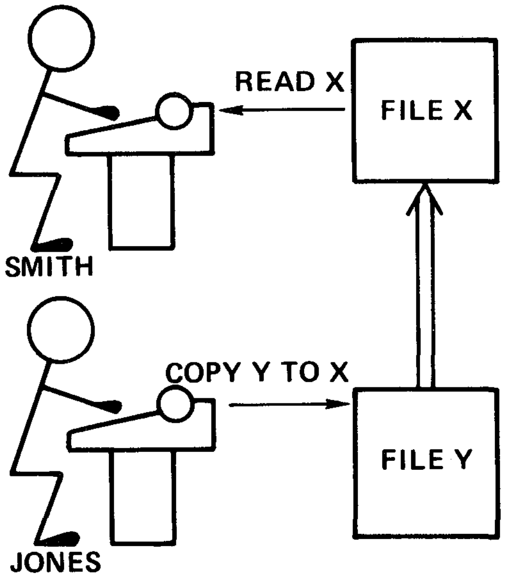
A _security policy_ is a data policy on _reading_ system objects. As such, it is specifically concerned with unauthorized disclosure of information.

A mandatory security policy is one in which the ability to read objects is administratively controlled. For instance, users might

IA-56831



ACCESS. Jones can be permitted to read file Y and write in file X; he has no access to file Z.



FLOW. Denied access to file Y, Smith gets confederate Jones to make a copy; flow controls could prevent this.

Figure 2. ACCESS CONTROL VS FLOW CONTROL

be assigned labels dependent on their titles (e.g., a personnel manager), and then only be allowed to read access to the object if they have the appropriate title.

Discretionary security policies tend to be more flexible, as shown in the owner example above.

The nature of a security compromise will depend on the data policy. An access control violation occurs when the system is unable to prevent data from being directly read by unauthorized users. A flow control violation occurs when the system cannot prevent information from being channeled from its original data object into one that can be read directly by an unauthorized user.

Note that the major threat addressed by this policy is compromise, or unauthorized disclosure, not sabotage--unauthorized modification (which probably occurs because those in the people-paper world assume all classified information is available elsewhere in triplicate). Sabotage has been described as an "integrity" problem, and is discussed by Biba [4].

DoD Policy

While laws concerning protection policies in computer systems throughout most of the Federal Government and in the commercial world are being debated, specific policies already exist within the national security community (DoD and the intelligence agencies), and indeed have even been unambiguously (i.e., mathematically) stated, or "modeled," so that conformance to the policies by computer systems can be more readily determined [5].

The DoD policy on data protection in automatic data processing systems is a mandatory security policy similar to the strict guidelines for the handling of classified papers ([6], [7], [8]). Here a major concern is on the dissemination of information--individuals should be allowed to access only the information for which they are cleared. In addition to the clearance level (Unclassified, Confidential, SECRET, TOP SECRET), an individual must have a need-to-know which may include approval for access to special categories or compartments of information. Together these form a partial ordering. Information is similarly labeled. (For purposes of this paper, an element of the partial ordering will be called the security level of the corresponding subject or object.) When multilevel information is to be processed concurrently in a computer system, two conditions which may be used to enforce the security policy are that in general:

1. A subject can read a data object only if the subject has a security level greater than or equal to that of the object;

2. A subject can write a data object only if the subject has a security level equal to that of the object. This condition is necessary to enforce a flow policy that would prevent information from being copied into a place (object) accessible by a subject with a lower security level.

In addition to the access and flow controls specific to data protection, a requirement exists for the auditing of protection-related events (described later in the document), and for the generation of labels on printed output stating the security level of the data.

Denial of Service

A denial of service condition results when a user is prevented from receiving the services to which he or she is entitled. It can be caused unintentionally if the operating system has a bug or if a user unknowingly introduces a bug into the system. It can be caused intentionally if the system was not designed to handle denial of service or if a malicious user introduces a bug into the system. For example, a user may be prevented from acquiring long-term storage space because it has been deliberately depleted, or execution time may be denied because the system "crashes." Alternatively, a user may be thwarted by more direct interference, such as having files deleted or modified in undesirable ways. A more insidious denial of service activity occurs when a malicious program masquerades as the normal service and causes a user to unknowingly reveal private information. (The masquerader might in addition perform the expected service, and thereby remain undetected.)

In order to counter the most general denial of service threat, the entire system must always operate correctly--applications software as well as any utility assistance from the operating system can never make mistakes or cause errors that will hamper the user from completing the task at hand. Because techniques for verifying the correctness of arbitrary programs are not yet available, addressing the general denial of service threat will be exceedingly difficult. However, an operating system should be able to defend against attacks involving resource exhaustion and masquerading. For instance, the protection policy might specify, "A subject can only create new objects if a system-maintained quota for the subject is not spent." Thus, if the quota is exhausted, the subject will be unable to exhaust another user's resources.

Mechanism

"Mechanism" refers to the features of a computer system that together enforce the protection policy. These features of the computer system may include algorithms, data bases, and protection hardware. To be effective they must be complete, correct, and self-protecting.

Nibaldi [9] describes an approach to building a computer system to maximize the likelihood that the resulting system is faithful to the policy. The approach requires identifying all protection-related functions, segregating them from the rest of the computer system functions, and then isolating them to prevent tampering. The result is called a "Trusted Computing Base" (TCB). It is a consequence of computer security research that culminated in security kernel technology [10]. As the protection-critical portion of the operating system, it should be completely able to mediate access to services independently of other software, and above all, be verifiable.

It must be noted that a TCB is defined as hardware (including firmware and microcode) as well as software. A number of hardware features have been identified that not only allow simpler software (potentially easing verification), but also expedite access mediation such as virtual segmented memory, capabilities, and user I/O. Hardware mechanisms are discussed by Tangney [11].

Four categories of protection mechanisms are considered: prevention mechanisms, detection mechanisms, recovery mechanisms, and mechanisms to support operations and maintenance.

Prevention

Prevention mechanisms actively work to implement the policy and prevent breaches. The following areas are of particular importance:

Data protection refers to the mechanisms that directly implement the relevant data (both access and flow) and authorization policies.

System integrity refers to the ability of the operating system or TCB to maintain its own integrity by protecting itself from tampering. It includes the ability to protect users from each other by providing virtual environments.

Denial of service mechanisms act to prevent denial of service attacks through the operating system.

Authentication mechanisms are needed so the system can reference the appropriate authorizations.* This area also includes mechanisms that allow subjects to authenticate that they are dealing with the system (as opposed to a masquerader), with specific objects, and with other subjects.

Confinement describes a condition of subjects in virtual isolation from other subjects and objects on the system. First identified by Lampson [12], confinement channels occur when the system resources are being shared. The reason they occur is that the operating system can signal information that is a direct result of resource utilization. If such "leakage" channels are not controlled, programs may use them to pass information in unauthorized ways, violating flow policy.

Two methods of passing information in this way are "storage" channels and "timing" channels. Storage channels involve shared control variables that can be influenced by a sender and read by a receiver, for instance when the information that the system disk is full is sent to a process trying to create a file. Timing channels also involve the use of resources, but here the exchange medium is time. For example, modulation of scheduling time can be used to pass information.

Storage channels can be detected using design verification techniques; timing channels are not easily detected because they depend on complex interactions of system and processes.

Detection

Detection mechanisms are passive policy enforcement devices. While the prevention mechanisms attempt to intercept potential violations, detection mechanisms monitor system activities, often maintaining records to aid in damage assessment, limitation, and recovery. Examples of detection mechanisms are time-of-use stamps, alarms, and audit trails.

It has been argued that if a policy violation can be detected, it might also be prevented. However, the detection mechanisms may

*The viability of the user authentication technique (e.g., passwords, fingerprints) may be difficult to measure. It is clear that if a password approach is used, the method for secreting passwords on the system (e.g., encryption) could be faulty and negate the approach. The method for judging the authentication approach will remain subjective until techniques for such calibration are developed.

include only very simple journaling programs that have no logic whatsoever regarding the significance of the events they record. Conceivably, certain violations could only be recognized after the fact, through complex logical and statistical analyses.

The primary modes of detection are auditing and surveillance.

Auditing is the practice of keeping records of system activities that have a bearing on the security of the system. Such activities include users logging in and out, the granting and revoking of access rights, and access violations, both attempted and successful (e.g., suspicious use of a storage channel). In order to monitor such activities, the system must be:

1. Designed so that critical actions are identifiable, and

2. Instrumented to follow these occurrences without seriously hampering the normal activities of the system.

To be effective, the detection apparatus (mechanisms and audit logs) must be secured, just as any other objects in the system. They would otherwise be a target for a penetrator trying to cover his tracks.

Surveillance is the active monitoring of the activities on the system in real-time. Surveillance facilities are especially useful to personnel in charge of security.

Recovery

Recovery mechanisms apply to the system components dedicated to restoring the secure state of the system in the event of an unexpected fault. Fault conditions may be induced by software (e.g., malicious user program) or hardware (failed component).

Software recovery may be impossible in some cases, for example, when a secret file has been read by an uncleared user. In such a case, there is no way to recover the compromised information. At best the software can recover a secure state in which other such violations would be prevented. However, where unauthorized modification is involved, the system software may provide backup capability.

Hardware recovery encompasses the broad area of fault tolerance and fault recovery. Fault tolerance implies that a system can sustain some amount of failure without propagating errors. The use of error correcting codes to negate the effects of one-bit errors in memory is one example of fault tolerance. Fault recovery extends the fault tolerance concept to the restoration of the system in the event of more extensive failures.

There is no known way of building the many components of a computer system so they will always work. It is more reasonable to assume that the system components will fail over some period of time. Consequently, provisions must be made to counter the effects of such a loss.

Simple detection of an erroneous situation is considerably more straightforward than the subsequent identification and isolation of the failed component for replacement or repair. The restart operation is particularly critical, because it implies some knowledge of the extent of error propagation which may be difficult to determine for every possible fault. It also implies that checkpoint or rollback states are preserved and are known to have survived the fault.

The most successful approaches to the general fault recovery problem have involved a number of redundant systems both for diagnosing and recovering from the fault before error propagation sets in. Methods of memory fault recovery might include, in the software area, diagnostic programs, cross checking programs (e.g., data base checksums), and subverter programs (which deliberately commit memory access violations to test access hardware).

Avizienis discusses fault recovery in more detail [13].

Operations and Maintenance

The operational and maintenance aspects fall in both the mechanism and assurance categories because they consist of procedures and programs that interplay to maintain the secure state of the system. This category includes startup, backup and restore, and configuration management procedures. Also included here are utilities for system control, such as initiating and changing authorizations, and setting quotas.

Assurance

Assurance features measure the degree of confidence that can be placed in the protection mechanisms, both hardware and software. Assurance can be legitimately gained through testing, but only after a system has been built. Yet while complete, exhaustive testing is possible for fairly small systems, it may be difficult, if not impossible, to determine if an arbitrarily large and complex operating system has been completely checked through the test process. As Dijkstra has noted [14], tests are only useful in determining the presence, not the absence, of bugs.

It is now accepted that care taken in the design and implementation of the system will increase one's confidence in that system. Also, developments in program verification will allow even greater assurance. In particular, in the TCB approach the formal methods can be applied to the TCB alone. If it indeed contains all protection-relevant functionality, correct design and implementation of the TCB provides an effective protection basis on which to build the remaining software in the system.

Automated aids facilitate the development and validation of operating systems by performing tasks with potentially fewer errors, at a faster rate, and with greater ease than a human could perform manually. Consequently, the use of such tools on a system will tend to heighten confidence in the system. Editors, compilers, and debuggers, for instance, are today considered absolutely necessary to the program development process. Automation is expanding in the validation area to include test case generation, automatic testing, and program verification. It must be emphasized that the advantage of automated tools lies in their support to a developer, not merely in their existence, unless the tool itself can be validated. Particularly in the case of testing and verification, the output of the tools should be "human-readable," to allow independent confirmation of the results.

Design

Certain elements of program design foster programs that not only lend themselves readily to testing, but also support eventual program verification. Among those elements are:

1. Top-down design, and

2. Design specifications.

Top-down design (also known as hierarchical design and stepwise refinement [15]) requires first, identifying major functions, and second, proceeding to identify the lesser functions that support the major ones. At each step, one specifies input, processing, and output, while avoiding implementation details. Top-down design forces one to carefully consider the implications of major design decisions at an early stage.

In such an approach it becomes useful to have graphical pictures of the levels, and a formal approach to documenting the interfaces between levels. A number of methodologies exist that incorporate special graphic techniques and formal design languages for recording design decisions in very precise specifications ([16], [17], [18]. See especially [16]).

Design specifications provide a high-level view of the behavior of the system. The top level specifications for an operating system or a TCB describe the user interface--what is visible to expand operating system or TCB software external to the TCB (i.e., the behavior of the high-level machine). If the top level specifications are written in a formal mathematical language, they may be syntactically and semantically checked, and analyzed for conformance to policy.

Lower level specifications aid top-down design by describing subsequent layers of abstract machine. Lower level specifications thereby facilitate the coding process and support program verification.

Implementation

The coding process can in itself increase the assurance level if clarity and readability of the programs is stressed. A beneficial side effect is more easily maintained programs. Top-down design and design specifications provide a medium for facilitating program implementation.

Three aspects of program implementation are noteworthy:

1. Modularity,

2. Abstract typing, and

3. Structured programming.

A modular program is one in which any logical portion can be changed without affecting the rest of the design. By keeping the modules fairly small (on the order of a printed page), they can be easier to write and debug; easier to maintain and change; and easier for a manager to control. But modular programming requires extra work, discipline, and may possibly cost more CPU time and memory space.

Modularity goes hand in hand with top-down design because the design may be structured in a hierarchy of modules--those at a higher level draw functionality from the modules at the next lower level.

Abstract typing is a concept that also draws on the module approach. A system may be designed such that the logic involving a particular type of object (e.g., I/O) is isolated in a special module with distinct interfaces. If other modules must manipulate an object of that type (in this case, an I/O device), they must invoke the appropriate type handler module. Implementation details can be hidden in a module; if they must be changed at some later time, the effect on the rest of the system will be minimized.

Structured programming is a philosophy of constructing programs in such a way that their logic is easily followed. It includes the concept of modularity, but also well-structured branching and control statements. In structured programs, all processing must consist of straight-line statements or function calls (where functions have a single entry and exit point), if-then-else statements, and looping constructs. (Extensions have been proposed, such as case statements, subroutines with multiple entries and exits, and restricted "goto.") Other aspects of structured programming are block structures with nesting for readability; maintenance of local variables that are never accessed from outside the module; and non-self-modifying code.

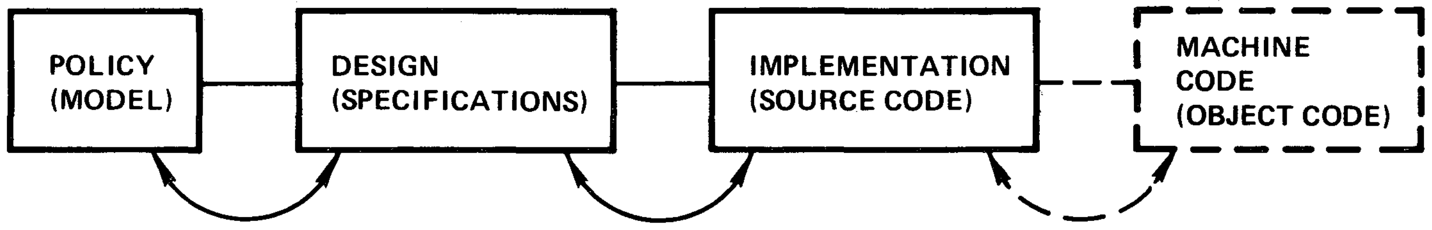
For the required constructs to be employed, they must be supported by the programming language. It would be difficult for assembly languages to support a structured programming style, although examples do exist, but many high-level languages can and do; e.g., Gypsy [18], Pascal [19], Modula [20], and Euclid [21].

One of Dijkstra's original objectives was that "mechanical proofs might be easier for a program expressed in some structured form" [15], referring to the machine-processable format that structured programs present. Verification systems currently under development in fact depend on these characteristics of programs, and may restrict the programmer even more by, for example, forcing proper type matching and eliminating pointers.

Verification

At present it is possible to verify a design (described in a formal top level specification) by showing that it corresponds to a security model of DoD policy (this has been done manually for a small system, and verification facilities are under development that will treat larger ones automatically). It is also possible for implementations of small programs (in suitable, axiomatized programming languages) to be verified against their design specifications. The program is thus shown to implement the policy. The correspondence chain implied here is shown in figure 3.

IA-56832



The dashed box shows what may be informal auditing of the machine code for correspondence to the source.

Figure 3. CORRESPONDENCE CHAIN

Design verification has taken two forms: invariant and flow analysis. A proof of invariants shows that certain conditions hold, ie., as state transitions are made. Invariant analysis can be used to detect access control violations. Flow analysis detects if illegal information flows may occur with the design, i.e., if flow control violations may occur. Both invariant analysis and flow analysis are discussed by Millen [22].

Implementation verification demonstrates that a program is consistent with its design. For instance, assertions on the states of variables at specific points in the program are shown to hold for all possible inputs. The logic involved in implementation correspondence proofs is very direct. The problems arise in the enormous amount of processing required for even simple proofs. Also, not all programming conditions can be checked. A notable example is concurrency--dealing with multiprocessing and simultaneous events. A number of trustworthy operating systems are currently being planned and built to be processed by verification facilities.

Testing

Testing methods in general attempt to show that the expected events occur when expected inputs are presented. Exhaustive testing seeks to show that all possible events are handled, i.e., expected. The philosophy has often been that the user will not try to misuse the system by attempting unexpected operations. Penetration analyses test for flaws in the system that could be used to circumvent the protection controls. Penetrations were performed successfully in the early 1970s to demonstrate the seriousness of the computer security problem [10] and will continue to be used in the future. Test procedures are detailed by Yourdon [15].

SUPPORTING FACTORS

Factors which support the protection mechanisms by making them more amenable to users include human interface (how difficult it is to use the facilities), granularity of protected objects (defining the smallest or largest unit the system will protect), system sizing (amount of storage, number of terminals, etc., available to users), and computational speed (response time). These tend to be factors of functionality rather than of protection, but they nevertheless add a significant dimension to the evaluation criteria. It is expected that systems which fall within each level will be judged suitable for a given application based on such supporting factors.

SECTION 3

LEVELS OF PROTECTION

The evaluation factors have been configured into seven levels, each of which identifies an increased degree of internal protection. The detailed descriptions include the technical, observable features of operating systems upon which an evaluation could be based. To summarize briefly:

- At level 0, there is no basis for confidence in the system's ability to protect information.
- At level 1, recognition of some attempt to control access is given, but only limited confidence in the viability of the controls is indicated.
- At level 2, minimal requirements on the protection policy must be satisfied; assurance is derived primarily from attention to protection during system design and extensive testing.
- At level 3, additional confidence is gained through methodical construction of the protection-related software components of the operating system (i.e., the TCB implementation), and modern programming techniques.
- At level 4, formal methods are employed to verify the design of the TCB implementation.
- At level 5, formal methods are employed to verify the software implementation of the design.
- At level 6, object code is analyzed and the hardware support is strengthened.

The levels of protection are ordered such that a system ranked at one level also qualifies for a lower level. For example, a level 3 system would be at least as strong as a level 2 system. Even though a system may exhibit elements from several different levels, it will be evaluated at the highest level for which it satisfies all the requirements.

In a number of instances a level might be attained in more than one way. For example, a design verification need not follow a specific methodology deemed appropriate; a comparable methodology will be equally as effective.

LEVEL 0: NO PROTECTION

A level 0 designation implies <u>null</u> capability, and would initially be applied to all unevaluated systems. In many instances of older operating systems, there are no, or only incomplete, provisions for protecting information from unauthorized access. Even the most general form of access control, limited access to the operating system via user-id and password, may not be required by the system. Where it is assumed that the environment in which the system runs is "benign," the lack of even minimal precautions is understandable. For example, even though a diverse collection of users might operate on the system (such as on a computer used for research projects at a university), users would not be expected to have malicious intent. As a consequence, the system is at most designed to protect against gross carelessness (e.g., in writing a file tagged read-only), not against a determined subverter (who might change the tag, then write).

In summary, there is no assurance that the system can restrict users to some subset of the total information and services available. A level 0 categorization indicates there is no evidence that the system will adequately protect information.

LEVEL 1: LIMITED CONTROLLED SHARING

The level 1 evaluation is a recognition of the presence of credible data access controls capable of providing minimal protection. Designers have seriously begun to address the problems of controlled information sharing in some more recently developed time-sharing operating systems.

Protection Policy

A protection policy of enforcing access control to data objects is expected at this level. Protection policies will likely follow the discretionary model--individuals are allowed to reference the information objects only in certain ways, which may be determined by labels or tags associated with both subjects and objects. The policy may also include algorithms for determining when a user can change the authorizations to a given object.

Specific Protection Mechanisms

The specific mechanisms which enforce the protection policy provide operating system protection (isolation) and user virtual spaces. Mechanisms to enforce a data policy on access control are provided. System access is gained through specific login subsystems that require a user attribute (e.g., finger print) or information only an authorized user should have (e.g., password).

No special protection-related detection requirements are made on systems at this level; however, as a performance or economic measure, accounting subsystems may measure the activity on the system.

No special fault-tolerant hardware is assumed. However, software diagnostics should attempt to detect errors that could hurt the system either by making it temporarily unavailable (inaccessible) for repair, or by destroying information stored on primary and secondary storage media. Backup and restore utilities and procedures exist for recovering file systems in the event of a fault.

Assurance

No particular standards for the operating system development are required at this level, although it is expected that what has been loosely referred to as "best commercial practice," or "good engineering practice," is followed. Confidence in the system is measured by code inspections and by the results of "industry-standard testing" (general debugging and tests of functionality).

Residual Risk

A level 1 system is only assumed to allow reasonable access control. Consequently, the flow control required by DoD policy may be non-existent. The operating system cannot be assumed to protect information on the basis of labeling; hence, either this cannot be a prerequisite or the application must provide the labeling capability from the protection that is provided by the operating system. Even this must be done with care since the operating system is capable of negating controls in the applications.

Summary

Although protection is not of major importance to the design, the system does have some limited means of controlling access. Testing is the only means by which the protection mechanisms are validated. The essential elements of level 1 systems are listed in figure 4. It is likely that many commercially available operating systems released within the last few years would be categorized at this level.

POLICY	MECHANISM	ASSURANCE
Some form of discretionary or mandatory protection	● PREVENTION	●DESIGN
	Data Protection Access Control	Good Engineering Practices
	System Integrity Isolated operating system Per-process virtual environment	●IMPLEMENTATION
		Inspections
		●VERIFICATION
	Authentication System/user (login)	●TESTING
	● DETECTION	Production Testing Debugging Functional Testing
	● RECOVERY - HARDWARE	
	Software Fault Detection Diagnostics	
	● OPERATIONS/MAINTENANCE	
	Backup/Recovery	

Figure 4

Level 1: Limited Controlled Sharing

LEVEL 2: EXTENSIVE MANDATORY SECURITY

The concern at level 2 is that the protection policy accommodate extensive mandatory security. Within the computer system, this means that

1. Authorizations to read data can be administratively controlled;

2. Flow controls prevent the data from being compromised; and

3. The integrity of the data can be maintained through write access controls.

The national security community has applications in which mandatory security is essential if more than one clearance level of user, or more than one level of data classification, may be present on the system at one time. Due to operational necessity, this can often be the case. At this level, in addition to satisfying the requirements of levels 0 and 1, the operating system acts in accordance with DoD policy.

Protection Policy

The system should support mandatory security control over and above any discretionary authorization policies.

Specific Protection Mechanisms

The specific protection mechanisms which foster an operating system of this level contribute to the enforcement of the protection policy and to the prevention of certain classes of denial of service attacks. Typically, in order to enforce the mandatory policy stored on computer systems, users, their processes, and information objects are labeled appropriately by the operating system. These labels must be protected across operating system actions.

Denial of service is addressed by implementing some form of "time-slice" scheduling policy, preventing any one user or program from effectively locking out all others from the CPU. Denial of service by the operating system is also accomplished if any user/process action can cause the system to "crash;" the possibility of such actions occurring should be minimized. The masquerading problem is addressed by mechanisms allowing the user to authenticate the system (e.g., by killing all currently active processes and initiating a new login).

Protection should be extended to consideration for information after it leaves the confines of the computer system: printouts, punched cards, and other forms of output must be labeled appropriately.

Specific protection-oriented actions will be audited, or recorded, in order that suspicious and incriminating actions might be detected--even if not prevented. Specifically, violations, output production, time of access, and time of login/logout should be recorded.

Fault detection in hardware should focus on protection-related hardware mechanisms (e.g., by using the software subverter approach).

Assurance

Confidence in the system is spurred by the techniques used to develop the system, namely modern programming practices. Structured programming techniques promote the writing of understandable code--code which is consequently more easily debugged. However, extensive testing is relied on for assurance. Penetration testing--testing in which attempts are made to exploit errors in the system and subvert the policy--is extensive.

Residual Risk

Although extensively tested, a level 2 system is still subject to design and coding errors. Testing should detect any obvious flaws; yet subtle ones might linger, to the advantage of untrusted users who are in a position to exploit them.

Summary

Level 2 systems support a mandatory security policy. Some attention is given to preventing denial of service by the operating system, and there is an attempt to audit, or record, certain protection-related events. Extensive testing, including penetration analyses, are relied on for assurance. A few systems modified for high-integrity DoD applications are expected to fall in this category. The essential elements of level 2 systems are listed in figure 5.

POLICY	MECHANISM	ASSURANCE
	• PREVENTION	•DESIGN
Discretionary	Denial of Service	•IMPLEMENTATION
	Time-slicing	
Mandatory Security	Masquerading	Modern Programming Techniques
Denial of Service	Authentication	
	User/System	
		•VERIFICATION
	• DETECTION	•TESTING
	Audit Loggin	Penetrations/Hardening
	Violations (attempted/ successful)	
	Classified Output Production	
	Time-of-use	
	Logins	
	• RECOVERY - HARDWARE	
	Software Fault Detection	
	Subverter programs	
	• OPERATIONS/MAINTENANCE	
	Output Labeling	

Figure 5

Level 2: Extensive Mandatory Security

LEVEL 3: STRUCTURED PROTECTION MECHANISM

It is at level 3 that the focus on high integrity protection mechanisms intensifies. At level 2, confidence that the mechanisms implement the protection policy is derived from careful adherence to methodological approaches to developing the protection-related functions of the operating system.

The hardware and software that perform these functions comprise a trusted computing base (TCB). The TCB has direct responsibility for the protection of the system. Not only is the TCB to be more carefully designed and implemented with respect to protection, it is not dependent on other software, and can protect itself from tampering. (The functionality required here has been documented by the author [9].)

Mechanisms that attempt to provide the protection needed for safe information sharing are built directly into the system, rather than added onto it.

Protection Policy

There is no change in policy from level 2.

Specific Protection Mechanisms

The specific protection mechanisms that contribute to a level 3 system all relate to clearly identifying and isolating the TCB of the system that will have the responsibility for enforcing the protection policies. Key to this ideal are mechanisms that permit complete mediation of all accesses to information objects, and isolation of the TCB itself for protection. The hardware, for example, may provide for segmented memory and specific protection on each segment. The TCB need only control the setting of protection modes, and the hardware will automatically check for invalid accesses. This kind of protection could, of course, also apply to the TCB code and data, providing the necessary isolation.

Assurance

By appropriately structuring the software that implements the protection features of a system, one can achieve more easily designed, coded, debugged, and maintained software. The methodologies that aid software development employ top-down design, abstract types, and structured programming in a high-level language.

Visibility into the design is gained by top-level specifications, providing a high-level description of the external interface to the TCB. Such a description of the external behavior of the TCB aids the testing process by delineating specific test cases. The kinds of testing required for level 2 acceptance will still be necessary here.

Residual Risk

Level 3 systems, by their construction, may invite greater confidence than level 2 systems. However, the testing process is still the main source of assurance; consequently, level 3 systems carry the same type of residual risk as is found in level 2 systems.

Summary

Protection is extremely important to the design of level 3 systems. Protection mechanisms are identified, isolated, and made independent of other software, allowing for ease of informal verification and analysis. Assurances go beyond testing because there is a methodological and structured approach to the design of the software involved in protection. But testing is still the primary means of assurance. The testing process is, however, aided by high-level descriptions of the user interface (e.g., top level specifications). The essential elements of level 3 systems are listed in figure 6.

POLICY	MECHANISM	ASSURANCE
Discretionary	• PREVENTION	•DESIGN
Mandatory Security	System Integrity	Structured Methodology
Denial of Service	Isolated protection mechanism	Top-down design
	Complete mediation	Top level design described in a "design" or "specification" language
	•DETECTION	
		•IMPLEMENTATION
	•RECOVERY - HARDWARE	Structured programming
	•OPERATIONS/MAINTENANCE	•VERIFICATION
		•TESTING Based on TLS

Figure 6

Level 3: Structured Protection Mechanism

LEVEL 4: DESIGN CORRESPONDENCE

The main distinction to be made for systems at this level is that formal methods are employed to confirm trustworthiness from the design. At this level, mathematical proofs of correspondence of the design to a security policy, represented by a security model, are required.

Protection Policy

There is no change in protection policy requirements from level 3.

Specific Protection Mechanisms

A specific requirement of the system is that it be able to audit the use of storage channels. These channels might be detected as a result of the formal verification techniques or by penetration analysis; however, they may not be easily removed without affecting the system in an adverse way. By imposing restrictions on the way resources are being shared, the system may no longer be allowed to use an optimal algorithm for resource utilization. The use of such channels can be detected with auditing mechanisms, and the information obtained from the auditing mechanisms can be analyzed later to find the source and seriousness of the channels' exploitation.

Hardware failures become increasingly more critical at level 4 as more confidence can be gained from the software implementation. At this level, it is required that the system be able to crash "softly"--restart (at some checkpoint location) with data in a consistent state--in the face of hardware errors, with support for recovery.

Assurance

Whereas the mechanisms used to enforce the protection policy may be addressed even in level 3 systems, additional assurance is sought at level 4. The additional assurance is that which comes from the completeness advantages of mathematically supported design verification. At level 3, insight into the overall design should be provided by top-level specifications of the external interfaces. At level 4, the specifications are required to be in a form that proves the design corresponds to an accepted security model. Both invariant and flow analyses are required.

However, a "correct design" does not imply a correct implementation. The source and machine code must be verified to correspond to the design as described in the specification (either through compiler verification or by some other means) for complete assurance. This form of verification will only be required at higher levels.

Even with formal design verification, functional testing is still needed. In addition to that required at previous levels, test cases are also required and are obtained from the specifications.

Configuration management becomes especially important at this level because the verified specification is expected to correspond to the implemented design. Changes should be controlled and audited. Also, because of the likelihood that requirements on the system may change, reverification procedures must be established. These might well be an extension of normal configuration management procedures.

Residual Risk

By undergoing rigorous design verification, level 4 systems are less likely to suffer from subtle design errors that may result in information flow through covert leakage channels. However, that the design is correctly implemented is not guaranteed.

Summary

Assurances extend to proofs of design-to-model correspondence through formal verification, showing that the design obeys an approved model of DoD policy. All identified leakage channels are audited. A number of systems under development for DoD are expected to fall into this category. The essential elements of level 4 systems are listed in figure 7.

POLICY	MECHANISM	ASSURANCE
Discretionary	• PREVENTION	• DESIGN
Mandatory Security		Formal, top level specifications (TLS)
Denial of Service		
	• DETECTION	• IMPLEMENTATION
	Audit Logging Leakage Channels	
		• VERIFICATION
	• RECOVERY - HARDWARE	Design-to-model proof Flow analysis Invariants
	H/W Fault Tolerance Limited operations	
		• TESTING
	• OPERATIONS/MAINTENANCE	Test Case Generation From TLS
	Configuration Management Reverification	

Figure 7

Level 4: Design Correspondence

LEVEL 5: IMPLEMENTATION CORRESPONDENCE

In level 5 systems, the implemented system must be shown to formally correspond to the verified top-level design. Also at this level, more stringent requirements for denial of service provisions, hardware fault tolerance, and leakage channel control are demanded.

Protection Policy

Additional policy matters to be considered involve the denial of service aspects--those involving the right of authorized users to an equitable share of all the resources of the system, not just the use of the CPU. No formal model of denial of service protection for the consideration of formal verification exists. Validation of conformance to policy in that respect must come about through extensive testing.

Specific Protection Mechanisms

The prevention of extensive exploitation of the covert leakage channels must be provided at this level. In particular, storage and timing channels, identified through design verification and testing, must be narrowed to limits that conform to the perceived threat. The exploitation of any known channels should be monitored through the use of on-line, real-time, surveillance tools.

Space resources (e.g., based on priority) are equitably allocated at this level.

Hardware-supplied backup systems and redundant circuits aid the fault-tolerance required of the hardware at this level. The unpredictability of hardware failures and the potential results necessitate the support that can be gained in addition to software.

Assurance

The importance of this level rests soundly on the proof of implementation, shown either by direct correspondence to a security model (in which case the design embodied in the implementation would be shown to correspond), or by correspondence to a design previously shown to correspond to the security model.

Proofs of correspondence, while possible to produce manually, may be automated, at least for fairly simple programs. However, proof of a code-to-design correspondence, even for simple programs, requires the specification of the system in more detail than a top-level specification might show. The provision of lower level specifications may be necessary as intermediate steps to the proof process. In addition, the source programs must be written in a language suitable for verification, and, at present, assertions must be added.

Penetration analyses are focused on identifying information leakage channels (such as timing channels) that might not be addressed by verification.

As yet, no control of the compilation phases has been required, although visual inspection of generated source and assembly code should satisfy one that no "trap doors" or "Trojan horses" have been implanted to circumvent the verified protection controls.

Residual Risk

Level 5 systems have the advantage of extensive program verification. At this stage, software ceases to be a weakness of the system. Hardware becomes more of a threat, even with extensive fault tolerance capability.

Summary

At this level, the state-of-the-art (and somewhat beyond) in computer security is brought to bear on the development of the protection-related software. Verification extends not only to proofs of correspondence of design to model but also to proofs that the implementation faithfully carries out the design. At this level, stringent requirements are made on the hardware (through backup systems) to decrease the probability of security breach due to hardware failure. All identified leakage channels are narrowed to tolerable limits. The essential elements of level 5 systems are listed in figure 7.

POLICY	MECHANISM	ASSURANCE
Denial of Service	●PREVENTION Denial of Service Space Quotas Collusion Timing channels bandwidth-limited Storage channels bandwidth-limited ●DETECTION Real-time surveillance tools ●RECOVERY - HARDWARE H/W Fault Recovery Backup systems ●OPERATIONS/MAINTENANCE	● DESIGN Low level specifications (LLS) ● IMPLEMENTATION Verifiable Implementation Strongly typed language Assertions ● VERIFICATION Code-to-design proofs ● TESTING Test Case Generation From LLS Penetration analyses Timing channels

Figure 8

Level 5: Implementation Correspondence

LEVEL 6: OBJECT CODE ANALYSIS

At this, the final currently defined stage, the last measure of reassurance is provided in the form of an analysis of compiler output, ie., object code. A proof of correspondence of object code to security model is indicated, (and thus satisfies the verification requirements for levels 4 and 5). However, a check of generated machine code against source code verified to correspond to a proven design would suffice.

Hardware requirements tighten here, too, as the probability of failure shifts from software to hardware. Although the impact of hardware fault should be softened as the result of provisions made at lower protection levels, formal approaches to understanding the behavior of hardware must be attempted here.

Protection Policy

No change to the operative protection policy is necessary at this level. Assurances that the system behaves in conformance with the protection policy is now extended to the object code and hardware.

Specific Protection Mechanisms

At this stage, the emphasis is on hardware mechanisms, for the software has undergone extensive verification. Fault handling must move from fault detection and fault tolerance to fault recovery.

Assurance

Assurance gained at this level comes from the careful analysis of the generated object code. That this code fulfills the requirements of the security model is one aspect that must be ascertained.

In addition, the bare machine must be more carefully verified if it is to support the programs that are also so thoroughly verified. This kind of understanding comes from interface specifications of the hardware, as is done for the TCB, from which formal statements can be made about the behavior of the security-relevant hardware under certain circumstances (e.g., changes in physical environment). Test case generation should follow from the hardware interface specifications.

Residual Risk

Level 6 systems offer a degree of confidence which is only imaginable from today's technology. Any threats at this level would be a result of highly improbable hardware errors, or, more likely, a failure in the personnel, administrative, physical, or communications security provisions.

Summary

At level 6, formal analysis of the object code produced by the compiler is required. Axiomatization of the underlying hardware base, and formal verification of the security-relevant hardware mechanisms, are also required. It is recognized, however, that these requirements are beyond the anticipated state-of-the-art of verification in the 1980s. The essential elements of level 6 systems are listed in figure 8.

POLICY	MECHANISM	ASSURANCE
Denial of Service	• PREVENTION	• DESIGN Hardware specifications (HWS)
	• DETECTION	• IMPLEMENTATION
	• RECOVERY - HARDWARE Fault Recovery Self-diagnosis/correction	• VERIFICATION Object code-to-source code proof HWS analyzed against TLS
	• OPERATIONS/MAINTENANCE	• TESTING Test Case Generation From HWS

Figure 9

Level 6: Object Code Analysis

SECTION 4

CONCLUSION

The sheer volume or criticality of applications that run on computer systems now, and of those that will run in the coming decade, necessitate careful attention to protection-related issues in the design of operating systems. Systems that purport to handle such information transactions will be more in demand; consequently, a means of determining their acceptability will be required. This report documents criteria for the evaluation of operating systems in which a TCB methodically designed, implemented, tested, and verified ranks highly. The reason for this is, of course, the recognition that ad hoc techniques of system development, no matter how cleverly implemented, cannot offer the assurance of methodical confirmation of the implementation.

The criteria, as stated, attempt to cover all known threats and the approaches to combating them. To allow the criteria to accommodate innovation, and to remain flexible in the face of change, certain precautions have been taken in the establishment of these criteria. Care has been taken to avoid specifying the mode or vehicle of implementation (e.g., hardware or software). Instead, attention has been focused on functionality--what must be accomplished to combat an abridgement of the relevant protection policy. Due to the ordering of protection levels, as the requirements are made more stringent, responses to newly perceived threats may be added as additional levels.

REFERENCES

1. Lee, T. M. P., P. Neumann, G. J. Popek, P. S. Tasker, S. T. Walker, C. Weissman, "Processors, Operating Systems and Nearby Peripherals: A Consensus Report," in "Secure Operating System Technology Papers for the Seminar on the DoD Computer Security Initiative Program," NBS Special Publication, Gaithersburg, Md, 17-18 July 1979.

2. Denning, D. E. and P. J. Denning, "Data Security," ACM Computing Surveys, Volume 11, Number 3, September 1979, pp. 227-249.

3. DeMillo, R. A., D. P. Dobkin, A. K. Jones, and R. J. Lipton, (ed.), Foundations of Secure Computation, Academic Press, 1978.

4. Biba, K. J., "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, The MITRE Corporation, Bedford, Mass., June 1975.

5. Bell, D. E. and L. J. LaPadula, "Secure Computer Systems," ESD-TR-73-278, Volume I-III, The MITRE Corporation, Bedford, Mass., November 1973-June 1974.

6. Department of Defense Directive 5200.28, "Security Requirements for Automatic Data Processing (ADP) Systems," December 18, 1972 (including Change 2, 29 April 1979).

7. Department of Defense Manual 5200.28-M, "ADP Security Manual," January 1973 (including Change 1, June 25, 1979).

8. Department of Defense Regulation 5200.1-R, "Information Security, Program Regulation," December 1978.

9. Nibaldi, G. H., "Specification of a Trusted Computing Base," M79-228, The MITRE Corporation, Bedford, Mass.

10. "Computer Security Developments Summary," MCI-75-1, Electronic Systems Division (AFSC), L. G. Hanscom Field, Bedford, Mass., December 1974.

11. Tangney, J. D., "Minicomputer Architectures for Effective Security Kernel Implementation," ESD-TR-78-170, The MITRE Corporation, Bedford, Mass., October 1978.

REFERENCES (Concluded)

12. Lampson, Butler, "A Note on the Confinement Problem," CACM, 16:10, October 1973, pp. 613-615.

13. Avizienis, A., "Fault-Tolerant Systems," IEEE Transactions on Computers, Volume C-25, Number 12, December 1976.

14. Dijkstra, E. W., "The Structure of THE-Multiprogramming System," Communications of the ACM, May 1968, pp. 341-346.

15. Yourdon, E., Techniques of Program Structure and Design, (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1975).

16. Jones, C. "A Survey of Programming Design and Specification Techniques," IBM, Santa Teresa Laboratory, San Jose, Ca. 95150.

17. Neumann, P. G., "Computer System Security Evaluation," Proc. NCC, January 1978.

18. Good, Donald I., R. M. Cohen, C. G. Hook, L. W. Hunter, D. F. Hare, "Report on the Language Gypsy: Version 2.0," ICSCA-CMP-10, University of Texas at Austin, Department of Computer Science, September 1978.

19. Hoare, C. A. R., and N. Wirth, "An Axiomatic Definition of the Programming Language PASCAL," Acta Informatica, Volume 2, pp. 335-355, 1973.

20. Wirth, N., "Modula: A Language for Modular Multiprogramming," Software Practice and Experience, Vol. 7, 1977, pp. 3-35.

21. Lampson, B. W. et al., "Report on the Programming Language Euclid," SIGPLAN Notices, Volume 12, Number 2, February 1977.

22. Millen, J. K., "Operating System Security Verification," M79-223, The MITRE Corporation, Bedford, Mass., September 1979.