DESIGNING THE GEMSOS SECURITY KERNEL FOR SECURITY AND PERFORMANCE*

Dr. Roger R. Schell
Dr. Tien F. Tao
Mark Heckman

Gemini Computers, Incorporated
P. O. Box 222417
Carmel, California 93922

INTRODUCTION

Gemini Computers, Inc. offers as a commercial product a family of secure, high performance computer systems based on the Intel iAPX 286 microprocessor. These systems are designed to meet the Class B3 requirements of the DoD Trusted Computer System Evaluation Criteria, [1] and a developmental evaluation by the DoD Computer Security Center is ongoing. An earlier paper [2] of about a year ago discussed the major concepts underlying the design and the functionality of the Gemini Multiprocessing Secure Operating System (GEMSOS). The security kernel as discussed in that paper is structured into eleven distinct layers, and as of that time only the lower five layers were implemented. All of the layers described have now been implemented and delivered as a Version 0 kernel, and a production Version 1 is currently being implemented. The purpose of this paper is to report on the major design choices for security functionality and to report the results of initial system performance measurements on the Version 0 GEMSOS commercial product.

BACKGROUND

The GEMSOS security kernel design is based on the trusted computer system technology that has emerged over the past decade. This technology provides a high degree of assurance that a system developed in accordance with the principles underlying the DoD Trusted Computer System Evaluation Criteria can be objectively evaluated to determine its ability to protect sensitive information from unauthorized viewing or modification. The primary experience with this technology is with general-purpose operating systems for single processor computers.

The Gemini design extends this security technology into the realm of a multiprocessor computer specifically intended for incorporation as a component of embedded systems. The question is often raised whether secure computers can be expected to deliver high performance. In the design of the GEMSOS security kernel, a good deal of attention has been given to supporting throughput and response time without adverse impact on security assurance. Of particular importance are the operating system techniques provided to ensure that the multiple processors can indeed be used effectively to increase overall system performance in concurrent computing applications.

The Reference Monitor Foundation

The reference monitor is the primary abstraction for dealing with a system that is designed to be "evaluable" with respect to security, i.e., a system for which we want to have a high degree of assurance of its correct security behavior. In this abstraction a system is considered as a set of active entities called "subjects" and a set of passive entities called "objects". The reference monitor is the abstraction for the control over the relationships between subjects and objects and for the manager of the physical resources of a system. To be effective in providing security, the implementation of a reference monitor must be: (1) tamper-proof, (2) always invoked, and (3) simple enough for analysis. The hardware and software that implement a reference monitor that meets these principles is defined as a security kernel. [3]

Security Policy Model

For a specific set of applications, e.g., for DoD systems, there will be a particularization of the reference monitor abstraction that incorporates the "security policy" (the desired security behavior) of the system. This particularization is formally defined in a "security policy model." The choice of a model will be influenced by a desire to have an intuitive tie to the engineering properties of the target system. Thus, in selecting a model for a trusted computer, it is desirable that the model's objects can easily represent the security-relevant information repositories of the contemplated applications. By far, the most widely used formal security policy model is the Bell-LaPadula model. [4] This model has a level of abstraction that is high enough to permit application to a wide variety of specific designs and is also deliberately designed to be extended to support system-specific policy refinements. This model has been used as the basis of the GEMSOS security kernel design.

An Extensible TCB

A pivotal concept in developing a system that is secure in a practical sense is the identification of a Trusted Computing Base (TCB). A security policy model is expected to model a broad range of actual systems. However, if the system is of a practical size it is expected that it will be too

---

*This is a reprint of a paper published in 1985. The text and figures have been scanned from the original document. Original citation:

R. R. Schell, T. F. Tao, and M. Heckman, "Designing the GEMSOS security kernel for security and performance." in *Proceedings of the 8th National Computer Security Conference*, Gaithersburg, MD, 1985, pp. 108-119.

complex to systematically evaluate for security. However, the reference monitor concept provides a basis for identifying a small and simple subset of the system that is responsible for and able to assure the security of the total system. This subset is called the TCB. The TCB includes both (1) the security kernel that implements the reference monitor and manages the physical resources, and (2) the "trusted subjects" that support refinements to the fundamental policy supported by the security kernel.

As noted in a recent paper by Marvin Schaefer of the DoD Computer Security Center, [5] if the TCB has a strict hierarchical layering it is possible to extend a mandatory policy security kernel to support a richer set of security properties, such as those desired for a discretionary security policy of a particular application. The GEMSOS security kernel has the kind of strict layering that was postulated. The Intel iAPX 286 processor that is used in the Gemini computer provides four strictly hierarchical, hardware enforced protection rings that enforce the strict layering. In particular, the most privileged ring (Ring 0) is devoted to the mandatory policy security kernel. Typically, Ring 1 would be similarly devoted to the particular discretionary policy of an application.

The key to the evaluation of the security of a TCB is the Descriptive Top Level Specification (DTLS) and for the Class A1 a corresponding Formal Top Level Specification (FTLS) for the interface to the TCB. For the GEMSOS security kernel the approach is to have specifications that are themselves layered so that there is a DTLS of the mandatory security kernel as well as a DTLS for the refinements, such as that for discretionary security policy; the latter specification includes references to the underlying mandatory DTLS. Thus, the GEMSOS security kernel provides an ideal foundation for a truly extensible TCB for support to a wide range of extended policies, such as those frequently encountered in military embedded systems.

MAJOR SECURITY CONCEPTS

A reference monitor implementation such as the GEMSOS security kernel must mediate all access by active entities (called "subjects") to passive entities (called "objects"). The GEMSOS kernel permits or prevents each access by a subject to an object based on relationships between the subject's authorization and the object's sensitivity. This set of relationships is called a security policy. The GEMSOS mandatory policy security kernel can enforce any of the various policies that are represented by the lattice of security labels used in the Bell and LaPadula mathematical security model. [4] This model provides a set of rules for controlling the dissemination and modification of information in a secure system. Kernel calls map into the rules of the model, and internal data structures represent the model's mathematical sets. The GEMSOS security kernel is secure because it is a valid interpretation of this security model. In the sections that follow, we define and introduce the major security system concepts and then use those concepts to describe the specific GEMSOS security kernel features.

Subjects and Objects

Rules in the Bell and LaPadula model are concerned with controlling the access of subjects to objects. *Subjects* are "active" entities that observe or modify objects. *Objects* are "passive" entities that are observed or modified. A subject is defined as a process executing in a specific domain and may be thought of as a (process, domain) pair. Objects are distinct, logical entities that contain information and possess security attributes called access classes.

Domains

A *domain* is a set of objects to which a subject has a given type of access, e.g., the "observe domain." Domains in the GEMSOS security kernel are determined by the hardware-enforced ring mechanism. The rings define a set of hierarchically ordered domains (see the section on ring integrity).

Security Policy

The permission of "authorized" access and the prevention of "unauthorized" access by subjects to objects constitute the security enforced by a system. A *security policy* is the set of relations between subjects' authorizations and objects' sensitivities that determines permissible access. A system that enforces a particular security policy may be said to be secure only with respect to that policy. [3]

Nondiscretionary Security. A security policy based on externally defined constraints enforced by a secure system is called a "mandatory" or *nondiscretionary* security policy. A nondiscretionary policy controls all accesses by subjects to objects and may never be modified or bypassed within the system. The military classification and compartment policy is an example of a nondiscretionary policy.

Discretionary Security. Within the limits of mandatory controls, authorized subjects in the system may place additional constraints on other subjects' access to objects. This internally modifiable set of constraints is called a *discretionary* security policy. The military "need-to-know" policy is an example of a discretionary policy. The complete security of a system may include both mandatory and discretionary controls, but while discretionary controls provide finer access "granularity," in no case may they override mandatory controls. The GEMSOS mandatory policy security kernel (viz., Ring 0) enforces a nondiscretionary security policy and provides a base to which a discretionary policy may be added (e.g., in the Ring 1 "supervisor" domain, as noted above).

Access Classes

Every entity in the GEMSOS security kernel possesses an *access class*. The access class of an object reflects the object's sensitivity, viz., its "classification." The access class of a subject reflects the subject's authorizations to observe and modify objects, viz., its "clearance." The complete

nondiscretionary (mandatory) security attributes of any subject or object in the GEMSOS security kernel are defined by its access class.

Security Labels. *Security labels* are attached to every entity (subject and object) in the system. A security label is a representation of an entity's access class.

Access Components. An *access component* is a way of describing separately the secrecy and integrity security attributes that constitute an access class. Discussion of access components simplifies the description of relations between subjects and objects based on access classes (i.e., the nondiscretionary security policy). In the GEMSOS security kernel, an access component is defined similarly to a Bell and LaPadula "security level". Both the secrecy and integrity access components consist of two parts: a hierarchical classification level and a set of compartments (induced by disjoint "categories").

Access Component Dominance. One access component (A1) is said to *dominate* another access component (A2) if the hierarchical level of A1 is greater than or equal to that of A2, and A1's compartments are a superset of A2's compartments. The symbol "}=" is used in following sections to indicate dominance (e.g., "A1 dominates A2" is depicted as "A1 }= A2").

Dissemination and Modification of Information

The nondiscretionary security policy enforced by the GEMSOS security kernel addresses both the secure dissemination and secure modification of information. The access class of every subject and object in the GEMSOS security kernel contains an access component to control dissemination and another access component to control modification. These two access components are referred to as "secrecy" and "integrity," respectively.

Secrecy Protection

Secrecy protection in the GEMSOS security kernel is similar to the usual interpretations of "security" in the Bell and LaPadula model. [4] The notion of secrecy is concerned with the secure distribution of information. Although the exact statement of security in the model is considerably more complex, the rules for enforcing secrecy protection in the GEMSOS security kernel can be simply described by two properties:

1) If a subject has "observe" access to an object, the secrecy access component of the subject must dominate the secrecy access component of the object.
2) If a subject has "modify" access to an object, the secrecy access component of the object must dominate the secrecy access component of the subject.

Property 1 is called the "no-read-up" or "simple security" property. Its effect is to keep low-secrecy subjects from observing information of higher secrecy. Property 2 is called the "no-write-down" property or "*-property" (read "star-property"). Its purpose is to keep high-secrecy subjects from improperly transmitting sensitive information to low-secrecy subjects (e.g., with a Trojan Horse). Thus, a low-secrecy subject can never directly or indirectly observe high-secrecy information.

Integrity

The concept of *integrity* is concerned with the secure modification of information. The GEMSOS security kernel provides two complementary mechanisms for enforcing integrity: one in software and one in hardware. The software mechanism enforces an integrity policy equivalent to the strict integrity policy described by Biba.[6] The hardware mechanism supports Multics-like hierarchical protection Rings. [7] The integrity enforced by the ring mechanism is equivalent to the notion of "program integrity" [8] and is a subset of the strict integrity policy.

The two mechanisms for enforcing integrity are provided for efficiency reasons. Were no ring mechanism available, the complete strict integrity policy could still be enforced by separate processes (rather than separate rings) using the software mechanism, but the hardware ring switching mechanism is considerably faster than process switching.

Strict Integrity. Like secrecy protection, the rules for enforcing strict integrity protection in the GEMSOS security kernel can be described by two properties:

1) If a subject has "modify" access to an object then the integrity access component of the subject dominates the integrity access component of the object.
2) If a subject has "observe" access to an object then the integrity access component of the object dominates the integrity access component of the subject.

Property 1 is called the "simple integrity" property. Its purpose is to prevent subjects of low integrity from modifying objects of higher integrity. Property 2 is called the "integrity *-property." It prevents high-integrity subjects from observing and relying on information that a low-integrity subject might have modified. If high-integrity subjects could observe low integrity information then their behavior might be improperly influenced ("spoofed") by a low-integrity subject. The two integrity properties prevent low-integrity subjects from directly and indirectly modifying high-integrity information.

Ring Integrity. Under the GEMSOS ring integrity mechanism, each subject and object possesses a hierarchical ring level ranging from 1 (most privileged) to 3 (least privileged). Subjects are only permitted access (observe, modify, or both) to objects with equal or greater ring

numbers (equal or less-privileged rings). No access to objects with a lower ring number is permitted.

For example, only the GEMSOS security kernel supervisor is allowed to have ring level 1. No application program, therefore, can access any part of the supervisor, but the supervisor can access any object in rings 1 thru 3. Ring 0 of a Gemini system is reserved for the isolation and protection of the security kernel.

Trusted Subjects

In general, the properties of secrecy and integrity are strictly enforced by the GEMSOS security kernel. Rigid adherence to these properties, however, can complicate the use of a system by forcing extremely fine "granularity" of security on objects. For example, imagine a system where storage objects are large files, and which has an incoming stream of messages at different access classes. One logical method of distributing messages would be to have all incoming messages put in the same file, and to have a single process distribute the messages to the proper recipients.

The secrecy and integrity properties described above, however, force a secure system to create a different file for each access class in which to store messages. In addition, a different process of the appropriate class must be used to distribute messages out of each file. Distributing messages, a relatively simple operation on most systems, could become a needlessly complicated, resource-consuming procedure in a secure system.

Imagine instead that the secrecy and integrity *-properties were relaxed just for this application. All incoming messages could be temporarily put in the same file and a single process could distribute messages of any access class. What would be the security characteristics of the file and process? The file would have the most sensitive access class possible, since it could conceivably contain messages of that class. Messages in the file, however, could be of lower classes. The distributing process would need authorization to observe objects of the highest class in order to read the file, and be able to modify objects from the most sensitive class down to the least sensitive in order to distribute the messages at their appropriate class. Since the process would be operating at many different access classes simultaneously, it must be trusted not to improperly pass sensitive messages to subjects with insufficient authorizations.

In order to support this type of application, the Bell and LaPadula model includes a "trusted subject" as part of the model. Trusted subjects in the model are subjects unconstrained by the *-property. Trusted subjects in the GEMSOS security kernel, unlike Bell and LaPadula's definition, are trusted (the *-property is relaxed) only within a given range, and are therefore "multilevel" subjects instead of general trusted subjects. Even with this additional security constraint, the GEMSOS security kernel is still a valid interpretation of the Bell and LaPadula model for, as Bell and LaPadula write, "... restrictions of the concept of security will not require reproof of the properties already established because additional restrictions can only reduce the set of reachable states." [4]

GEMSOS SECURITY KERNEL FEATURES

Each of the structures and concepts described above is embodied in some way in the GEMSOS security kernel. The GEMSOS security kernel organization for implementation has been described previously, [2] and the reader interested in the specific primitives and kernel calls identified in the following discussions is encouraged to review this description. The following sections describe the major security features. The basic abstractions used in the GEMSOS design are segments, processes and devices. The segments are instances of "objects" of the model. The processes and devices are "subjects" of the model.

Security Labels

Security labels are representations of the sensitivity of objects and the authorizations of subjects. A security label is attached to every subject and object in the GEMSOS security kernel. Security labels are called "access classes" since they symbolize the complete set of security attributes possessed by each entity. The GEMSOS security kernel access classes (security labels) are records with two fields, representing secrecy and integrity access components. For secrecy, the default label has eight (8) hierarchical classifications and twenty-nine (29) nonhierarchical categories. For strict integrity, the default label has eight (8) hierarchical classifications and sixteen (16) non-hierarchical categories.

Some entities in the GEMSOS security kernel have two access classes: a maximum and a minimum. The secrecy and integrity access components of the maximum access class always dominate the secrecy and integrity access components of the minimum access class. The maximum and minimum classes together describe a range of permissible access classes.

Segments

All information in a Gemini system is contained in discreet, logical objects called *segments*. Each segment has an access class that reflects the sensitivity of information contained in the segment. Segments may be simultaneously and independently shared by multiple subjects but access to the segment (observe, modify, or both) on the part of each subject is controlled by the relationship between the segment's access class and each subject's access class, in accordance with the security properties of the model.

Every segment in the GEMSOS security kernel has a unique identifier. This identifier is effectively different for every segment ever created in any Gemini system. Unique identifiers are used internal to the kernel to prevent "spoofing" of the system by substituting one segment for another and are not visible at the kernel interface.

Eventcounts and Sequencers. The GEMSOS security kernel uses abstract data objects called "eventcounts" and "sequencers" for process synchronization and communication. These objects may be observed and modified by subjects so, to preserve security, the GEMSOS security kernel must control access to them. In order to identify eventcounts and sequencers, one eventcount and one sequencer are associated with the name of each segment. Each segment name, therefore, is used to identify an eventcount and a sequencer as well as a segment (see the section below on segment aliasing for more information on segment names). The eventcount and sequencer associated with a segment name have the same access class as the segment whose name they share. Process synchronization and communication is thus subject to the same rules of security as observing and modifying segments.

In order to modify an eventcount (using the primitive "advance") a process must be permitted modify access to the segment. Similarly, in order to observe an eventcount (using the primitives "read" or "await") a process must be permitted observe access to the segment. The primitive "ticket" for sequencers requires the potential for both observe and modify access to the segment. See the description by Reed [9] for more information on eventcounts and sequencers.

Volumes. Secondary storage in Gemini systems is divided into distinct logical volumes. Each segment is associated with only one volume, determined when the segment is created. Each volume may be considered to be a collection of segments. Volumes have two access classes, a maximum and a minimum, assigned when the volume is formatted. The secrecy and integrity components of the maximum access class must dominate the secrecy and integrity of the minimum class. The maximum and minimum volume access classes are upper and lower limits on the security of information contained on the volume (see the section below on the use of volumes).

Processes

A subject in the GEMSOS security kernel is a (process, domain) pair, where the domain is determined by the current hierarchical ring level at which the process is executing. The ring level determines the set of objects to which, within security constraints, the process potentially has access. Processes may change their ring levels, but the same process executing in a different ring is a different subject. The primary application of rings envisioned under the GEMSOS security kernel is for the creation of distinct domains so that a process can have up to three (rings 1, 2, and 3) "subjects." Each subject has a minimum and a maximum access class (security label) that is typically uniform for a process, no matter which ring it executes in. The secrecy and integrity access components of the maximum access class dominate those of the minimum access class. If the maximum and minimum access classes are equal then the subject is s "single-level" subject. If the two classes are unequal then the subject is a "multilevel" subject.

Single-level Subjects. Subjects that have equal maximum and minimum access classes are single-level subjects. Single-leve1 subjects may only have the access to objects permitted by the simple and "*" secrecy and integrity properties (see the sections on secrecy and integrity).

Multilevel Subjects. Multilevel subjects have unequal maximum and minimum access classes. This property of multilevel subjects gives them the ability to have both observe and modify access to objects whose access classes fall between the subject's minimum and maximum. Multilevel subjects are the GEMSOS security kernel implementation of "trusted subjects" (see the section on trusted subjects). Unlike general trusted subjects, however, multilevel subjects are only trusted within a range demarcated by their maximum and minimum access classes. Within this range, multilevel subjects are not constrained by the *-properties of secrecy and integrity (but they are still subject to ring integrity). Only subjects guaranteed not to improperly downgrade or modify information should be created as multilevel subjects.

I/O Devices

I/O devices are viewed by processes as system processes. Processes communicate with I/O devices using shared segments and may also use eventcounts. Like other processes, I/O devices have both maximum and minimum access classes. These limits are intended to reflect the security constraints imposed by the physical environments in which devices are located and are critical to the employment of a secure computer in a multilevel environment.

Devices may be either single-level or multilevel. Unlike processes, which are classified single-level or multilevel based on their minimum and maximum access classes, the Criteria [1] categorizes I/O devices as "single-level" or "multilevel" based on the access classes of the data they manipulate. Data transmitted or received by a single-level device has no attached security label. Single-level devices thus consider all data to have a single access class. Data transmitted or received by a multilevel device has a security label attached to or stored with the data in the same form as the data. Multilevel devices therefore may handle data with a range of access classes.

Single-level Devices. A single-level device handles data to which no explicit security label is attached. In many environments, the minimum and maximum access classes for a single-level device will be the same. A single-level device at a given time treats all input data as having a single access class, which is determined by the security of the physical environment at that time. Output data must have an access class that falls within the range of the device's maximum and minimum access classes.

In order to establish communication between a process and a single-level device in the GEMSOS security kernel (called "attaching" the device), the range of the subject must

"intersect" the range of the device. Specifically, the following relationships between the process's access classes and the device's access classes must hold to ensure that the process will be able to receive or send data through the device:

1) To receive ("read") information:

Process maximum secrecy }=
Device minimum secrecy

Device maximum integrity }=
Process minimum integrity

2) To send ("write") information:

Device maximum secrecy }=
Process minimum secrecy

Process maximum integrity }=
Device minimum integrity

An example of a single-level device with different minimum and maximum access classes is a log-on terminal in a room to which users with authorizations ranging from the highest possible (system-high) access class down to the lowest possible (system-low) access class have access. In this example, the maximum and minimum access classes of the device would be system-high and system-low respectively, although narrower ranges are possible in other situations. There is only the single terminal in the room and only one person is allowed in the room at a time. Users log on to the system through a "trusted path", [1] which allows the GEMSOS security kernel to directly and securely determine their access class; it then creates a process of the same access class to represent the user in the system.

After a user has logged on using the trusted path, and the user's process has attached the device, the GEMSOS security kernel considers the security of the terminal device environment to be the same as the security of the user's process. Different users will have different access classes, but at a given time there is only one user so the data has only a single access class.

When a single-level device receives data, an access class (security label) must be established for the data. If the current security of the device has been reliably transmitted to the GEMSOS security kernel (e.g., through a trusted path) then the attached process will have an access class that represents the current security of the device. The received data is usually assigned the maximum secrecy and minimum integrity of the process that attached the device. A single-level device with a range of access classes, as can be seen from this example, must have a trusted path of some sort in order to be used in a secure manner. If not, then the minimum and maximum access classes of the single-level device should be identical.

Multilevel Devices. Any data input or output through a multilevel device must have an access class that falls within the range defined by the device's maximum and minimum access classes. Multilevel devices may handle data with a range of access classes. All data transmitted or received by a multilevel device has a security label attached or stored along with the data.

In order for a process to attach (establish communication with) a multilevel I/O device, the following relationships between the process's access classes and the device's access classes must hold to ensure that the process can send and receive information through the device without violating security:

1) To receive ("read") information:

Process maximum secrecy }=
Device maximum secrecy

Device minimum integrity }=
Process minimum integrity

2) To send ("write") information:

Device minimum secrecy }=
Process minimum secrecy

Process maximum integrity }=
Device maximum Integrity

Process and Segment Interaction

This section explains how the GEMSOS security kernel controls the ability of subjects, viz., (process, domain) pairs, to access objects (segments). A process may create and destroy segments, may add segments to and delete segments from the process's address space, and may move segments between main and secondary storage.

The total set of objects to which a subject potentially has access is the subject's access domain. A subject's access domain is determined by the subject's hierarchical ring level. The subset of the access domain that includes all objects to which, at a given time, a subject actually has access is called the subject's *address space*. Segments are brought into a subject's address space using the kernel call "makeknown_segment".

Access Modes. The GEMSOS security kernel allows processes to have execute only, read-execute, read only, and read-write access mode combinations to segments. Of these access mode combinations, all but read-write are considered to be "observe" type access modes. Read-write is both an "observe" type and a "modify" type access mode. The GEMSOS security kernel has no write only (modify only type) access mode for segments.

A process may have only one access mode combination to a segment at a time, but may simultaneously have

different access modes to different segments. The access mode a process has to a segment is selected by the process at the time the segment is brought into the subject's address space. The actual type of access selected need only be a subset of the potential types of access allowed by the security policy. For example, if a process may potentially have both observe and modify type access to a segment based on the relationship between the process's and segment's access classes, it need not select the read-write access mode (observe and modify) to the segment, but can instead select any of the observe type access modes, since "observe" is a subset of its potential access types.

In order to have any of the observe access mode combinations to a segment, a process's maximum secrecy access component must dominate the segment's secrecy and the segment's integrity access component must dominate the process's minimum integrity. These requirements enforce the simple security and integrity "*" properties. In order to have modify access to a segment, the segment's secrecy access component must dominate the process's minimum secrecy and the process's maximum integrity access component must dominate the segment's integrity. These requirements enforce the secrecy "*" and simple integrity properties. In order to have both observe and modify access to a segment all four of these requirements must be met. Multilevel processes thus potentially have both observe and modify access to any segment whose access class falls within the process's range, while single-level processes are only permitted both observe and modify access to segments with the same access class as the process.

Segment Naming (Aliasing). In order to create or delete a segment, or to add a segment to its address space, a process must tell the GEMSOS security kernel the process-local name of the segment. The method of naming segments is called *aliasing*. Aliasing allows processes to uniquely identify shared segments in the system while still maintaining security.

A segment name consists of a system-wide component and a process-local component. The system-wide component is an index called an "entry number." A segment's entry number is the same for all processes, can be stored for future reference to the segment, and can be passed to and used by other processes. Were it not for the danger of covert information channels, the totally "flat" (non-hierarchical) entry number naming scheme would itself be sufficient for naming all segments.

Due to the danger of covert channels however, the system-wide entry number of a segment is always relative to a "mentor" segment. The mentor segment is identified by a process-local number that is not unique across processes, that cannot meaningfully be passed to another process, and that cannot be saved for future use (although the mentor itself also has a system-wide name, as described below). The paired process-local mentor segment number and system-wide entry number constitute a process-local segment alias used by a process for identifying the segment to the kernel.

A segment's alias is different from its ("invisible") unique identifier (described in the section on segments). Only one segment can have a particular (mentor, entry) pair as its name at a time, but if that segment is deleted then another segment can be created with the same name. The unique identifier of the new segment, however, will be different from the old segment.

A mentor segment may not be deleted until all segments named with that mentor have been deleted. This restriction prevents the problem of "zomb1e" segments that, although they exist in storage, cannot be accessed or deleted since they cannot be named.

Segment Naming Hierarchy. The process-local segment alias, viz., (mentor, entry) pair, of a segment can be used to add the segment to a process's address space. When a segment is entered into a process's address space, the process assigns the segment its own process-local segment number. The segment may then itself be used as a mentor segment if its process-local segment number is used as the mentor in the alias of another segment. Recursive application of this naming scheme results in a "hierarchy" of segment aliases.

Compatibility Property. The hierarchical segment naming scheme, using segment aliases, allows the GEMSOS security kernel to preserve system security by preventing the creation of covert channels through the use of segment names. Were processes able to name and thereby sense the presence or absence of any segment in the system (as they would in a flat naming scheme), this would constitute a source of covert channels. High secrecy level processes could signal low secrecy level processes, and low integrity level processes could signal high integrity level processes, just by creating and deleting segments.

The GEMSOS security kernel alias hierarchy, however, allows processes to name segments only where that naming will not cause a covert channel. The hierarchy prevents covert channels by strictly ordering the security relationship between segments and their mentors:

$$\text{Segment Secrecy} \} = \text{Mentor Secrecy}$$
$$\text{Mentor Integrity} \} = \text{Segment Integrity}$$

These properties of the hierarchy have been called the Compatibility Property [4] (for secrecy) and the Inverse Compatibility Property (for integrity). [6]

The compatibility and inverse compatibility properties are maintained because of the way segments are named. A segment name consists of a (mentor, entry) pair. The entry number is relative to that particular mentor and is considered to be information associated with the mentor segment. A

segment's entry number, as a result, is considered to have the same access class as the segment's mentor segment.

Whenever a process tries to create or delete a segment, or to add a segment to its address space, the process must "observe" the segment's entry number to see if, in fact, the segment exists. Since entry numbers are information associated with a mentor, a process must potentially have observe access to the segment's mentor in order to name the segment.

A mentor's alias consists of another (mentor, entry) pair, its mentor is named in the same fashion, and so on recursively back to the "root" of the naming hierarchy. Each segment's alias can thus be thought of as a vector consisting of a series of entry numbers that uniquely describes a "path" to the segment. Clearly, in order to name a segment, a process must be able to name every mentor segment in the segment's path and it must therefore potentially have observe access to each of the mentor segments.

If the compatibility property was not maintained and a segment's secrecy did not necessarily dominate the secrecy of its mentor, a situation might arise where a process that should have access to the segment based on the relationships between their access classes cannot have it, since the process cannot get observe access to the mentor. Every segment's secrecy, therefore, must dominate the secrecy of its mentor. Applying this rule recursively back to the root gives the result that secrecy is monotonically non-decreasing following a name path from the root to any segment.

Every segment has a unique path. This is achieved by requiring all segments with the same mentor to have unique entry numbers. Whenever a process creates or deletes a segment it specifies the name of the segment (conceptually "modifying" the segment's entry number to indicate the entry is in use or free). Since entry numbers are information associated with a mentor, a process must potentially have modify access to the mentor in order to create or delete a segment's name.

Because a segment name is a vector consisting of a series of entry numbers, changing any of the constituent entries would change the segment's name. A segment's name, therefore, must have at least the integrity of the segment (i.e., must dominate the segment's integrity). Since a segment's name consists of a (mentor, entry) pair, and since the entry number has the same access class as the mentor segment, the integrity of the mentor segment must dominate the integrity of the segment. Applying this rule recursively back to the root gives the result that integrity is monotonically non-increasing following a name path from the root to any segment.

To create a segment, a process requires the potential for both observe and modify access to the segment's mentor segment. Creating a segment affects only the segment's name, which is associated with the mentor, and does not affect the contents of, or information associated with, the segment being created. A process that creates a segment, therefore, does not need to be able to access the segment it creates. To delete a segment a process requires the potential for both observe and modify access to the segment's mentor and, in addition, requires the potential for observe access to the segment being deleted. This additional restriction is necessary because mentor segments may not be deleted. A process deleting a segment must therefore know if the segment it is trying to delete is a mentor segment, requiring the process to "observe" if any other segments are named using the segment as a mentor. Those segments names, as stated above, are information associated with the mentor and possess the same access class.

The Use of Volumes. Volumes are collections of segments useful for physically organizing and protecting information of similar access classes. Volumes are brought into the GEMSOS security kernel using the kernel call "mount_volume." The first time a volume is mounted, it is uniquely associated for the life of that volume with a segment that will serve as the "root" mentor to segments on the volume. The segment used as a volume mentor may serve as a volume mentor to that volume only, and must not ever have been a mentor to other segments before the initial mounting of the volume. In this way all segments on the volume are guaranteed to have unique pathnames, distinct from the pathnames of segments on other volumes. Were pathnames indistinct, the system could be spoofed by substituting one segment for another of the same name.

Volumes may be unmounted and mounted repeatedly, but, to ensure unique pathnames, if its mentor is deleted a volume may never be remounted. A volume whose mentor is deleted must be reformatted to be reused — a process that destroys whatever information the volume might contain. A volume mentor segment may not be deleted while the volume is mounted. A volume may not be unmounted if any of the segments on the volume are currently "known" in any process's address space. These precautions prevent the problem of "zombie" volumes (mounted volumes that cannot be unmounted because they are not addressable) and "orphan" segments (segments addressable but not able to be swapped-out to disk).

The names of all segments on a volume must have access classes that fall within the volume limits. Since some segments on the volume will have the volume mentor segment as their mentor, the volume mentor segment's access class must also fall within the maximum and minimum access classes of the volume. According to the compatibility property, secrecy is monotonically non-decreasing and integrity is monotonically non-increasing. When a volume is first mounted, therefore, the maximum potential secrecy access component of segments on the volume is the volume maximum, but the effective minimum secrecy access component becomes the mentor's secrecy. Similarly, the minimum possible integrity access component of segments on the volume is the volume minimum, but the

effective maximum possible integrity becomes the mentor's integrity.

In order to satisfy the compatibility property, the secrecy access component of any segment created on a volume must dominate the secrecy access component of the volume mentor and be dominated by the maximum secrecy of the volume. The integrity access component of any segment created on the volume must dominate the minimum integrity of the volume and be dominated by the mentor's integrity. These relationships are summarized below (where S indicates Secrecy, and I indicates Integrity):

S(volume max) }=
S (segment) }=
S (volume mentor segment)

I(volume mentor segment) }=
I(segment) }=
I (volume min)

To prevent a covert channel caused by the mounting and unmounting of volumes, a process that mounts or unmounts a volume must have a minimum secrecy access component dominated by the secrecy of the volume mentor and a maximum secrecy access component that dominates the maximum secrecy of the volume. The minimum integrity access component of the process must be dominated by the minimum integrity of the volume and the maximum integrity access component of the process must dominate the integrity of the volume mentor segment. That is:

S(process max) }=
S (volume max) }=
S (volume mentor segment) }=
S (process min)

I(process max) }=
I(volume mentor segment) }=
I(volume min) }=
I(process min)

An example of the use of volumes is a floppy diskette environment. Each floppy diskette is a different volume with its own maximum and minimum access classes. Disks are labeled with these classes at the time they are formatted. The first time a diskette is used, the process that represents the diskette user in the system creates a mentor segment for the diskette volume and mounts the volume. Other users' processes may share the volume if the security policy allows them to "make known" the volume mentor segment in their address spaces. After the volume is unmounted and the diskette put away, the diskette may be reused so long as its mentor segment is not deleted. If the diskette volume's mentor is deleted, the diskette must be reformatted to be reused, destroying any information contained on the diskette.

SYSTEM PERFORMANCE MEASUREMENTS

The security-kernel approach to the design of a multilevel secure computer system offers a solution to the size and complexity problems that have dogged other approaches. However, some previous implementations of security kernels have resulted in systems with discouraging performance, reportedly as much as 75 to 90 percent below that of equivalent non-trusted systems.

Design Factors

After about a decade of work in the area of trusted systems, substantial information is available on factors that relate to some of the disappointing performance that has been experienced. Several of these factors are discussed below.

Language Efficiency. For verification purposes, security kernels are written in high-level languages which are chosen for features such as strong typing of data. Some of these languages tend to produce inefficient code, In the case of the GEMSOS security kernel, the PASCAL language has been chosen because of its support for evaluation. The compiler used is not particularly efficient, but is considered typical for microcomputer compilers. Thus there is some performance impact from choosing to use a higher-order language, but no significant additional impact from the choice to support security.

Hardware Support. The different security classes of users and information must be distinguished and, where incompatible, kept separate; when hardware support is inadequate, the supporting overhead restricts the bandwidth of information that a secure computer system can process. Previous work has identified [3] four general architectural areas where hardware features are particularly useful or necessary: process management and switching, memory segmentation, input/output mediation, and execution domains. The Intel iAPX 286 processor used for the Gemini computers provides a high level of hardware support in all these areas.

System Architecture. The implementation choices for organizing the internal structure and the environment for applications have s major impact on the performance of any operating system. Generally the approach in the GEMSOS security kernel has been to take advantage of the techniques found effective in the industry as long as these do not adversely impact security. For example the choice of system computational model, e.g., process oriented or capability based, can directly affect system response time. We have chosen the proven process oriented approach, as reflected in the security discussion above. Two other choices of particular importance relate to the how multiprocessing is implemented, viz., techniques for avoiding bus contention and for preventing the kernel from being a critical section bottleneck. These are discussed below.

Bus contention is a potential performance concern in the Gemini multiprocessor configuration, since all processors share a single bus. In reality however, only shared, writable segments need be in a global memory on the shared bus. All other segments can be in processor-local memory. Our use of a purely virtual, segmented memory permits the kernel to determine exactly which are the shared, writable segments. The memory manager layer internal to the kernel totally controls the allocation to global memory to insure that only the required segments are in global memory. This policy can require some transfer between local and global memory but this structure markedly controls bus contention by allocating segments to the processor-local memory whenever possible. Our experience with sample applications is encouraging in that typically much less than 10% of the references of a processor are to a global memory. Thus, a number of processors can be effectively used on a single, shared bus.

In most, if not all, previous security kernel implementations, the kernel is a single critical section. This means that the kernel can be executed by only a single process at a time, and in addition cannot be interrupted. For a single processor, the adverse impact of this choice is somewhat contained in that there are no other processors that can be forced to wait. However, even in the single processor case, real time response may be affected because there will be no response to an interrupt from an external device until any call to the kernel that has begun before the interrupt is completed. The GEMSOS security kernel is designed to be close to interruptible throughout its entire execution.

The impact of the critical section design choice is much more severe in the multiprocessor case. With a critical section, if one processor is executing in the kernel when another processor wants to invoke the kernel, the second processor must wait in essentially an "idle" condition until the first processor completes its execution of the kernel call. The degradation is clearly a function of the amount of service, viz., the number and type of calls that the application demands of the kernel. In addition, the degradation increases as the number of processors increases. In the GEMSOS security kernel, there are limited critical sections internal to the kernel itself that can result in contention, but the kernel itself is not a critical section so that multiple processors can execute simultaneously in the kernel.

Performance Results

Although the design approach to provide good performance is of interest, the real proof of any system is the actual measured performance. We have taken some preliminary measurements that focus on demonstrating (1) the throughput performance for multiprocessor configurations and (2) the response to realtime inputs. It is emphasized that these measurements were taken on Version 0 and that several performance enhancements have been designed for Version 1 that have not yet been implemented. Although we believe the results illustrate the general behavior, these early measurements should not be considered a definitive characterization of the Gemini product.

Multiprocessing Throughput. We have prepared a message processing emulation that runs on the multiprocessor environment. The emulation is reminiscent of a military communications processing application. The processing consists of a "front end processor" servicing multiple communication lines, and interfacing to an additional single communication line. The demonstration is not connected to physical communication lines, but emulates receiving messages from an input buffer segment and putting output messages in an output buffer segment.

The details of the demonstration are not very important to the measurements, but will be briefly summarized. Messages are treated as a series of line blocks of 84 bytes each. Each message requires some amount of processing and then a resulting message is placed in the message queue segment for "transmission" through the output buffer segment. A message processing process is created for each pair of input communication lines; this process does all the message processing and places the message, a line block at a time, in the message queue. In addition, there is a single output process that takes messages, a line block at a time, and puts them in the output buffer. The kernel synchronization primitives for eventcounts are used to ensure that each message processing process waits for room in the message queue and to ensure that the output process takes the messages from the queue when they are available.

All the processes and buffer segments are actually created and used. The only actual input/output is to a screen for interface to the test operator. A display is generated for each message line block and, at the end of the "test run", timing information based on the internal real-time clock is displayed. The demonstration is intended to show the additional capacity that can be provided by additional processors. A distinct "test run" is used for each processor configuration of from one to seven processors, emulating service for from two to fourteen input lines, with each processor servicing two lines. The test operator selects a parameter that controls the amount of processing for each message. This parameter determines the amount of processing that is in the demonstration application versus the amount of processing in the kernel. The processing is simulated by repeated execution of a mix of instructions taken from a communication processing application.

An estimate is made of the percentage of the total processing time spent in the application for various choices for the parameter that controls the amount of processing for each message. The measurements taken on a single processor are used to normalize the results for additional processors, so that the number of "effective processors" can be determined. The number of effective processors becomes the primary figure of merit for the true effectiveness of the multiprocessing. Because of the multiprocessor contention within the kernel, this will be reduced if the application

requires extensive services from the kernel, viz., as the percentage of application processing is reduced.

The results of a series of actual measurements as described above are summarized in Figure 1. This shows that, for substantial application processing, there is a nearly linear increase in system throughput as the number of processors is increased. This clearly reflects that there is very little contention between processors for the shared bus. Furthermore, even when only 85% of the processing is in the application, there is still effectively six processors worth of throughput for a seven processor configuration.

Real Time Response. We have prepared s set of tests that require real-time response to external input. For this test we process communications input in a way that requires character-at-a-time processing. For each character there is an interrupt and the system must respond before two additional characters are received, or else with the hardware used for the interface, the communication will be broken. This is not necessarily the preferred implementation for such communication, but serves as a useful test implementation. Thus, if the interrupts occur frequently (i.e., for a high transmission rate), it is essential that the kernel be interruptible. The specific test is an implementation of the HDLC support for the X.25 protocol that is used for the Defense Data Network interface to a computer host. The test has no higher level "flow control" protocol, so the input data must be received in real time.

The communication is synchronous, so that the actual amount of time available to respond to the interrupt depends on the transmission rate used. The other primary parameter for the test is the size of the HDLC frame used. Each HDLC frame includes about three bytes of overhead in addition to the frame size, so that the effective throughput will be inherently reduced for small frame sizes. The test uses a kernel call for each HDLC frame. Thus the choice of frame size affects both the probability that the interrupts will occur while the kernel is executing and the amount of kernel processing required for each frame.

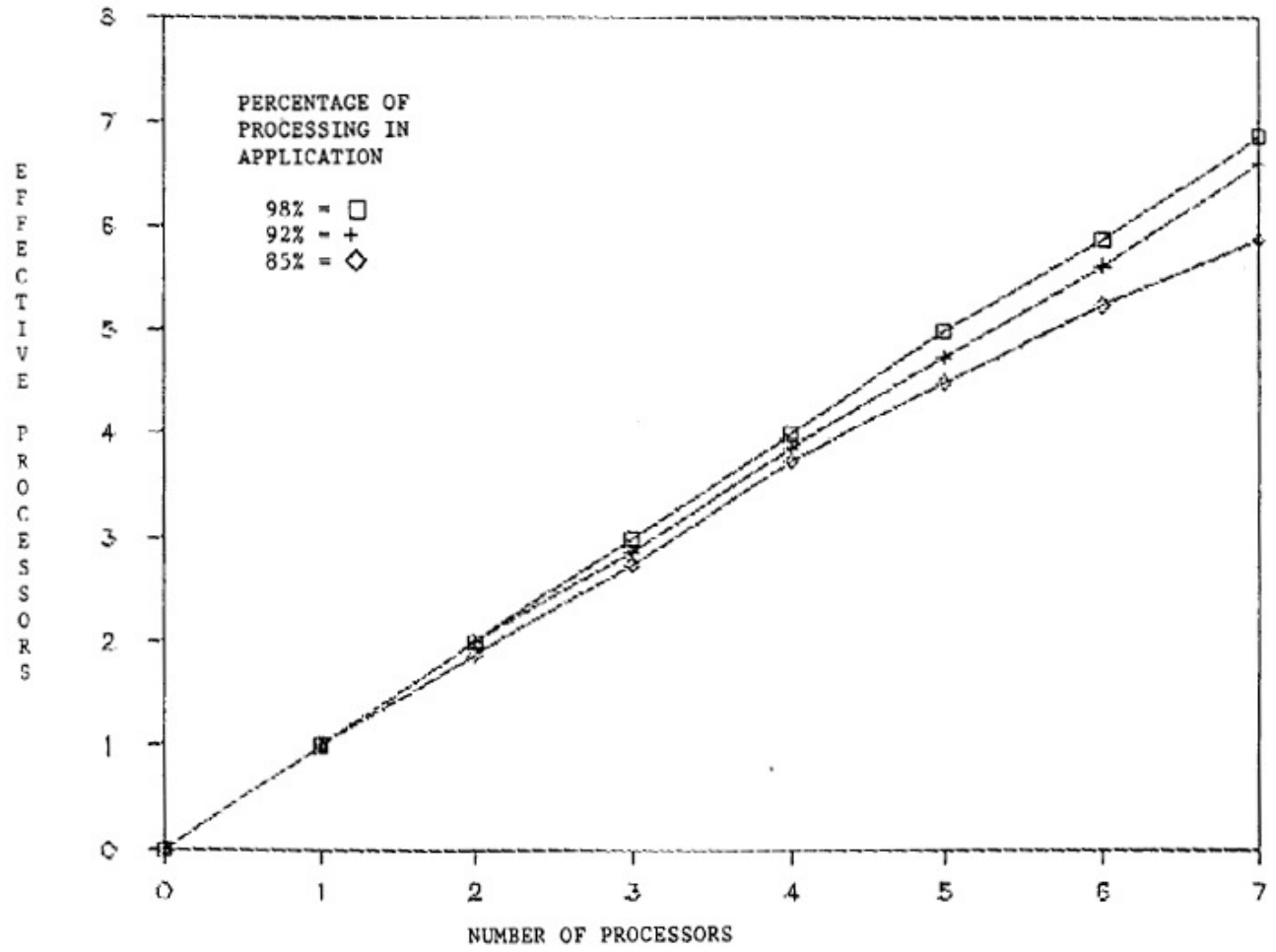

Figure 1 - Gemini Multiprocessor Enhancement

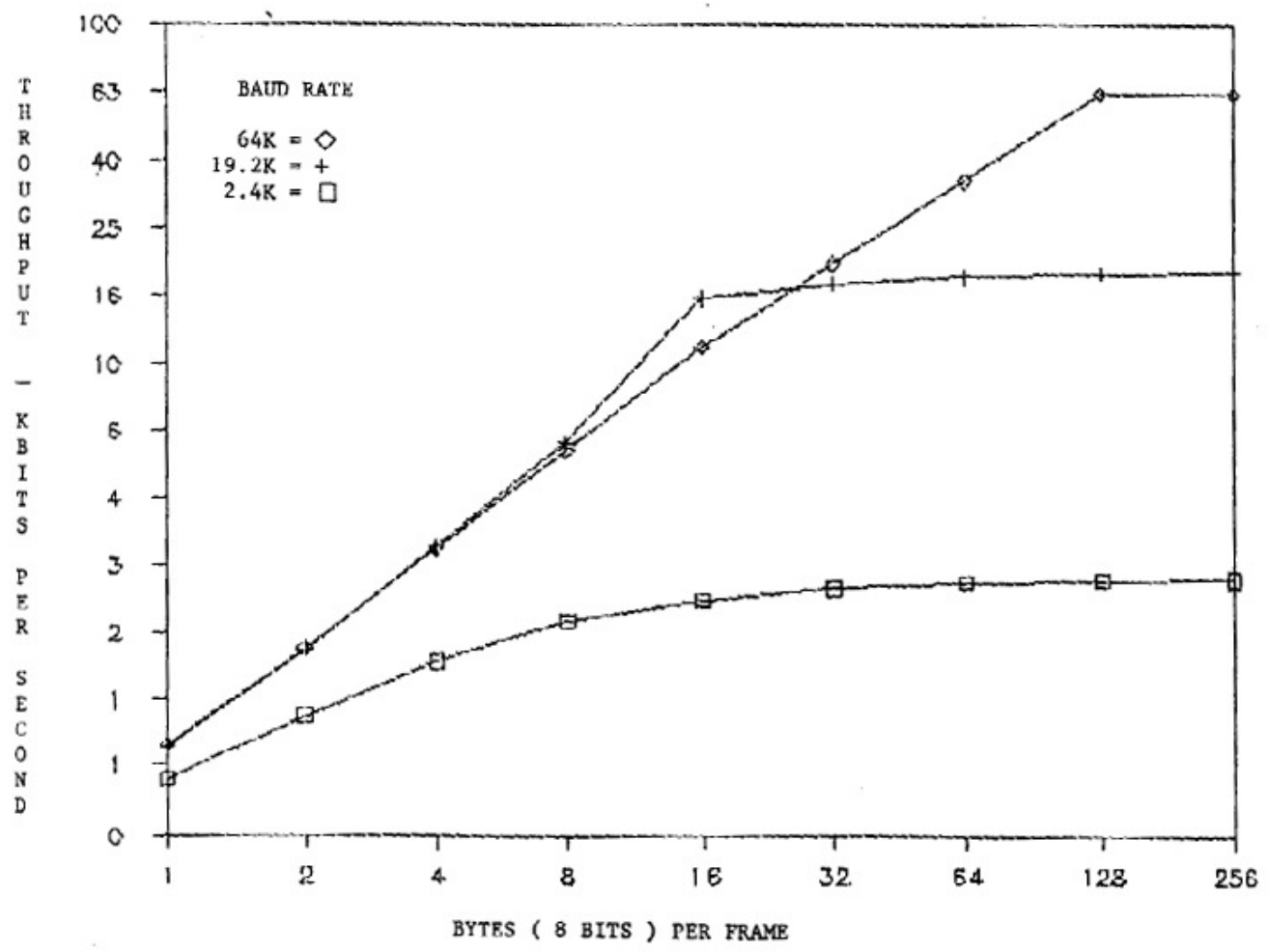

Figure 2 - Gemini Real-Time Processing

The test is conducted using two Gemini computers connected with an HDLC link. The application programs in one computer makes a series of kernel calls to transmit a sequence of frames and the other makes a series of kernel calls to receive the sequence of calls. Figure 2 shows the results of the series of tests. For all the tests there were no communications errors, demonstrating that the kernel was able to support the real-time response at all the transmission rates tested — up to 64 kilobits per second. The amount of application and kernel processing per frame was constant for all the data points. For purposes of this test, unrealistically small frame sizes are included to illustrate that, even when the throughput is being limited by this processing, the` character-by-character real-time response is still maintained

SUMMARY AND CONCLUSIONS

The design of the GEMSOS security kernel for the Gemini commercial product has been influenced first by the Class B3 security requirements and second by the objective of high performance. We have described the major security design choices and believe that these provide an implementation that is particularly attractive for application in embedded systems. We have also reported the results of preliminary throughput and real-time performance measurements. These show effective real-time capability and nearly linear increase in throughput as the number of processors is increased up through seven processors. Although these are on an early version of the kernel that will be improved, we believe that these results already demonstrate that a secure system can also have high performance.

REFERENCES

1. DoD Trusted Computer System Evaluation Criteria, CSC-STD-0O1-83, 15 August 1983, DoD Computer Security Center, Ft. Meade, MD.

2. Schell, R.R., and Tao, T.F., Microcomputer-Based Trusted Systems for Communication and Workstation Applications, Proceedings of the 7th DoD/NBS Computer Security Initiative Conference, NBS, Gaithersburg, MD, 24-26 September 1984, pp. 277-290.

3. S.R. Ames, M. Gasser, and R.R. Schell, “An Introduction to the Principles of Security Kernel Design and Implementation “Computer, Vol. 16, No. 7, July 1983, pp. 14-22.

4. D.E. Bell and L.J. LaPadu1a, “Computer Security Model: Unified Exposition and Multics Interpretation,” Tech. report ESD-TR-75-306, AD A023588, The Mitre Corporation, Bedford, Mass., June 1975.

5. M. Schaefer and R.R. Schell, “Towards an Understanding of Extensible Architectures for Evaluated Trusted Computer System Products,” Proceedings of the 1984 Symposium on Security and Privacy, April 1984, pp. 42-49.

6. K.J. Biba, “Integrity Considerations for Secure Computer Systems,” Tech. Report ESD-TR-76-372, The Mitre Corporation, April 1977.

7. M.D. Schroeder and J.H. Seltzer, “A Hardware Architecture for Implementing Protection Rings,” Communications of the ACM, Vol. 15, No. 3, March 1972. PP. 115-124.

8. L.J. Shirley and R.R. Schell, “Mechanism Sufficiency Validation by Assignment,” Proceedings of the 1981 Symposium on Security and Privacy, IEEE, Cat. No. 81CH1629-5, April 1981.

9. D.P. Reed, and R.K. Kanodia, “Synchronization with Eventcounts and Sequencers,” Communications of the ACM, Vol. 22, No. 2, February 1979, pp. 115-124.