LEVEL II

M79-228

AD A108831

SPECIFICATION OF A TRUSTED COMPUTING BASE (TCB)

G. H. Nibaldi

30 November 1979

DTIC FILE COPY

235050

THE MITRE CORPORATION
Bedford, Massachusetts

DTIC ELECTE DEC 23 1981 S D

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

81 12 22 108

ABSTRACT

A Trusted Computing Base (TCB) is the totality of access control mechanisms for an operating system. A TCB should provide both a basic protection environment and the additional user services required for a trustworthy turnkey system. The basic protection environment is equivalent to that provided by a security kernel; the user services are analogous to the facilities provided by trusted processes in kernel-based systems. This report documents the performance, design, and development requirements for a TCB for a general-purpose operating system.

The information in this report is made available to stimulate technical discussion among industry and government personnel. The views and conclusions contained in this paper are those of the author and should not be interpreted as necessarily representing the official policies either expressed or implied of the Department of Defense or United States government.

This work was supported under Contract Number F19628-80-C-0001 as part of the DoD Computer Security Initiative Program.

TABLE OF CONTENTS

TABLE OF CONTENTS (Concluded)

SECTION 1

SCOPE

1.1 IDENTIFICATION

In any computer operating system that supports multiprograming and resource sharing, certain mechanisms can usually be identified as attempting to provide protection among users against unauthorized access to computer data. However, experience has shown that no matter how well-intentioned the developers, traditional methods of software design and production have failed to provide systems with adequate, verifiably correct protection mechanisms. We define a trusted computing base (TCB) to be the totality of access control mechanisms for an operating system. A TCB should provide both a basic protection environment and the additional user services required for a trustworthy turnkey system. The basic protection environment is equivalent to that provided by a security kernel;[1] the user services are analogous to the facilities provided by trusted processes in kernel-based systems.[2] This report documents the performance, design, and development requirements for a TCB for a general-purpose operating system.

In this report, there will be no attempt to specify how any particular aspect of a TCB must be implemented. Studies of present-day computer architectures [Smith 75, Tangney 78] indicate that in the near term a significant amount of software will be needed for protection regardless of any support provided by the underlying hardware. In future computer architectures, more of the TCB functions may be implemented in hardware or firmware. Examples of specific hardware or software implementations are given merely as illustrations, and are not meant to be requirements.

---

[1] A security kernel is a verifiable hardware/software mechanism that mediates access to information in a computer system. See [ESD 74], [Popek et al. 77], [KSOS 78], and [Schaefer et al. 77].

[2] Trusted processes are designed to provide services that could be incorporated in the kernel but are kept separate to simplify verification of both kernel and trusted processes. Trusted processes also have been referred to as "privileged," "responsible," "semi-trusted", and "non-kernel security-related (NKSR)" in various implementations.

This specification is limited to computer hardware and software protection mechanisms; not covered are the administrative, physical, personnel, communications, and other security measures that complement the internal computer security controls. For more information in those areas, see [DoD 5200.28], that describes the procedures for the Department of Defense.

1.2 DOCUMENT OUTLINE

The specification is organized in three additional parts. Section 2 lists the references. Section 3 contains the general design requirements for a trusted computing base. Detailed design requirements are found in section 4.

SECTION 2

APPLICABLE DOCUMENTS

The following documents form a part of this specification to the extent specified in this report. In the event of a conflict between the referenced documents and the contents of this specification, this specification shall be considered a superseding requirement.

2.1 GOVERNMENT DOCUMENTS

2.1.1 Directives, Manuals and Standards

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

b. Department of Defense Directive 5200.28, "Security Requirements for Automatic Data Processing (ADP) Systems," December 18, 1972 (including Change 2, April 29, 1978).

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

d. MIL-STD-483 Configuration Management

e. MIL-STD-490 Specification Practices

2.1.2 Reports

a. [Anderson 72] Anderson, J. P., "Computer Security Technology Planning Study," ESD-TR-73-51, Volume I, James P. Anderson & Co., Fort Washington, Pennsylvania (October 1972).

b. [Bell and LaPadula 73] Bell, D. E. and L. J. LaPadula, "Secure Computer Systems," ESD-TR-73-278, Volume I-III, The MITRE Corporation, Bedford, Ma. (November 1973 - June 1974).

c. [Biba 75] Biba, K. J., "Integrity Considerations for Secure Computer Systems," ESD-TR-76-372, The MITRE Corporation, Bedford Ma. (June 1975).

d. [ESD 1974] "Computer Security Developments Summary," MCI-75-1, Electronics Systems Division (AFSC), L. G. Hanscom Field, Bedford, Ma., December 1974.

e. [Furtek 78] Furtek, Frederick C, "A Validation Technique for Computer Security Based on the Theory of Constraints," ESD-TR-78-182, The MITRE Corporation, Bedford, Ma. (December 1978).

f. [Good 78] Good, Donald, I., R. M. Cohen, C. G. Hock, L. W. Hunter, D. F. Hare, Report on the Language Gypsy: Version 2.0, ICSCA-CMP-10, The University of Texas at Austin, (September 1978).

g. [KSOS 78] KSOS Computer System Specification (Type A), WDL-TR-7808 Revision 1, Ford Aerospace Communications Corporation, Palo Alto, Ca., (July 1978).

h. [Lampson 73] Lampson, Butler, "A Note on the Confinement Problem," CACM, 10 (October 17), 613-615.

i. [Lipner 75] Lipner, Steven B., "A Comment on the Confinement Problem," MTP-167, The MITRE Corporation, Bedford, Ma. (November 1975).

j. [Nibaldi 79] Nibaldi, G. H., "Proposed Technical Evaluation Criteria for Trusted Computer Systems," M79-225, The MITRE Corporation, Bedford, Ma., (25 October 1979).

k. [Popek et al. 78] Popek, Gerald J., Mark Kampe Charles S. Kline, Allen Stoughton, Michael Urban, Evelyn J. Walton, "UCLA Data Secure UNIX - A Securable Operating System: Software Architecture," UCLA-ENG-7854, UCLA Computer Science Department, Los Angeles, Ca., (August 1978).

l. [Robinson et al. 77] Robinson, L., K. N. Levitt, P. G. Neumann, and A. K. Saxena, "A Formal Methodology for the Design of Operating System Software," in R. Y. Yeh (ed.), Current Trends in Programming Methodology, Vol. I: Software Specification and Design, Prentice-Hall, Englewood Cliffs, NJ, 1977, pp. 61-110.

m. [Saltzer 75] Saltzer, Jerome H, "The Protection of Information in Computer Systems," Proceedings of the IEEE, Vol. 63, No. 9 (September 1975).

n. [Schaefer et al. 77] Schaefer, Marvin, Barry Gold, Richard Linde, and John Scheid, "Program Confinement in KVM/370," ACM Annual Conference Proceedings, October 16-19, 1977, Seattle, pp. 404-410.

o. [Smith 75] Smith, L. "Architectures for Secure Computing Systems," ESD-TR-75-51, The MITRE Corporation, Bedford, Ma. (April 1975).

p. [Spec 78] AF Specification No. CP 0787796100D, Appendix 80: "Multilevel Security Requirements" (31 March 1978).

q. [Tangney 78] Tangney, John D., "Minicomputer Architectures for Effective Security Kernel Implementations," ESD-TR-78-170, The MITRE Corporation, Bedford, Ma. (October 1978).

SECTION 3

GENERAL REQUIREMENTS

3.1 SYSTEM DEFINITION

A TCB is a hardware and software access control mechanism that establishes a protection environment to control the sharing of information in computer systems.[3] A TCB is an implementation of a reference monitor, as defined in [Anderson 72], that controls when and how data is accessed.

In general, a TCB must enforce a given protection policy describing the conditions under which information and system resources can be made available to the users of the system. Protection policies address such problems as undesirable disclosure and destructive modification of information in the system, and harm to the functioning of the system resulting in the denial of service to authorized users.

Proof that the TCB will indeed enforce the relevant protection policy can only be provided through a formal, methodological approach to TCB design and verification, an example of which is discussed below. Because the TCB consists of all the security-related mechanisms, proof of its validity implies the remainder of the system will perform correctly with respect to the policy.

Ideally, in an implementation, policy and mechanism can be kept separate so as to make the protection mechanisms flexible and amenable to different environments, e.g., military, banking, or medical applications. The advantage here is that a change in or reinterpretation of the required policy need not result in rewriting or reverifying the TCB.

In the following sections, general requirements for TCB design and verification are discussed.

---

[3] Under hardware and software we include implementations of computer architectures in firmware or microcode.

3.2 PROTECTION POLICY

The primary requirement on a TCB is that it support a well-defined protection policy. The precise policy will be largely application and organization dependent. Four specific protection policies are listed below as examples around which TCBs may be designed. All are fairly general purpose, and when used in combination, would satisfy the needs of most applications, although they do not specifically address the denial of service threat. The policies are ordered by their concern either with the viewing of information--security policies--or with information modification--integrity policies; and by whether the ability to access information is externally predetermined--mandatory policies--or controlled by the possessor of the information--discretionary policies:

1. mandatory security (used by the Department of Defense--see [DoD 5200.28]), to address the compromise of information involving national security;

2. discretionary security (commonly found in general purpose computer systems today);

3. mandatory integrity; and

4. discretionary integrity policy.

In each of these cases, "protection attributes" are associated with the protectable entities, or "objects" (computer resources such as files and peripheral devices that contain the data of interest), and with the users of these entities (e.g., users, processes), referred to as subjects. In particular, for mandatory security policy, the attributes of subjects and objects will be referred to as "security levels". These attributes are used by the TCB to determine what accesses are valid. The nature of these attributes will depend on the applicable protection policy.

See Nibaldi [Nibaldi 75] for a general discussion on policy. See Biba [Biba 75] for a discussion of integrity.

3.3 REFERENCE MONITOR REQUIREMENTS

As stated above, a TCB is an implementation of a reference monitor. The predominant criteria for a sound reference monitor implementation are that it be

1. complete in its mediation of access to data and other computer resources;

2. self-protecting, free from interference and spurious modification; and

3. verifiable, constructed in a way that enables convincing demonstration of its correctness and infallibility.

3.3.1 Completeness

The requirement that a TCB mediate every access to data in the computer system is crucial. In particular, a TCB should mediate access to itself--its code and private data--thereby supporting the second criterion for self-protection. The implication is that on every action by subjects on objects, the TCB is invoked, either explicitly or implicitly, to determine the validity of the action with respect to the protection policy. This includes:

1. unmistakably identifying the subjects and objects and their protection attributes, and

2. making it impossible for the access checking to be circumvented.

In essence, the TCB must establish an environment that will simultaneously (a) partition the physical resources of the system (e.g., cycles, memory, devices, files) into "virtual" resources for each subject, and (b) cause certain activities performed by the subjects, such as referencing objects outside of their virtual space, to require TCB intervention.

3.3.1.1 Subject/Object Identification. What are the subjects and objects for a given system and how are they brought into the system and assigned protection attributes? In the people/paper world, people are clearly the subjects. In a computer, the process has commonly been taken as a subject in security kernel-based systems, and storage entities (e.g., records, files, and I/O devices) are usually considered the objects.[4] The precise breakdown for a given system will depend on the application. Complete identification of

---

[4]Note that a process might also behave as an object, for instance if another process sends it mail (writes it). Likewise, an I/O device might be considered to sometimes act as a subject, if it can access any area of memory in performing an operation. In any case, the policy rules governing subject/object interaction must always be obeyed.

subjects and objects within the computer system can only be assured if their creation, name association, and protection attribute assignment always take place under TCB control, and no subsequent manipulations on subjects and objects are allowed to change these attributes without TCB involvement. Certain issues remain, such as (a) how to associate individual users and the programs they run with subjects; and (b) how to associate all the entities that must be accessed on the system (i.e., the computer resources) with objects. TCB functions for this purpose are described in section 4, "Detailed Requirements".

3.3.1.2 Access Checking. How are the subjects constrained to invoke the TCB on every access to objects? Just as the TCB should be responsible for generating and unmistakably labelling every subject and object in the system, the TCB must also be the facility for enabling subjects to manipulate objects, for instance by forcing every fetch, store, or I/O instruction executed by non-TCB software to be "interpreted" by the TCB.

Hardware support for checking on memory accesses exists on several machines, and has been found to be very efficient. This support has taken the form of descriptor-based addressing: each process has a virtual space consisting of segments of physical memory that appear to the process to be connected. In fact, the segments may be scattered all over memory, and the virtual space may have holes in it where no segments are assigned. Whenever the process references a location, the hardware converts the "virtual address" into the name of a base register (holding the physical address of the start of the segment, the length of the segments, and the modes of access allowed on the segment), and an offset. The content of the base register is called a descriptor. The hardware can then abort if the form of reference (e.g., read, write) does not correspond to the valid access modes, if the offset exceeds the size of the segment, or if no segment has been "mapped" to that address. The software portion of the TCB need merely be responsible for setting up the descriptor registers based on one-time checks as to the legality of the mapping.

Access checking in I/O has been aided by hardware features in a variety of ways. In one line of computers, devices are manipulated through the virtual memory mechanism: a process accesses a device by referencing a virtual address that is subsequently changed by hardware into the physical address of the device. This form of I/O is referred to as "mapped I/O" [Tangney 78]. Other methods of checking I/O are discussed in section 4.

3.3.2 Self-Protection

Following the principle of economy of mechanism [Saltzer 75], the TCB ideally protects itself in the same way that it protects other objects, so the discussion on the completeness property applies here as well. In addition, not uncommonly many computer architectures provide for multiple protection "domains" of varying privilege (e.g., supervisor, user). Activities across domains are limited by the hardware so that software in the the more privileged domains might affect the operations in less privileged domains, but not necessarily vice versa. Also, software not executing in a privileged domain is restricted, again by the hardware, from using certain instructions, e.g., manipulate-descriptor-registers, set-privilege-bit, halt, and start-I/O. Generally only TCB software would run in the most privileged domain and rely on the hardware for its protection. (Of course, part of the TCB might run outside of that domain, e.g., as a trusted process.) Clearly, if in addition to the TCB, non-TCB or untrusted software were allowed to run in the privileged region, TCB controls could be subverted and the domain mechanism would be useless.

3.3.3 Verifiability

The responsibility given to the TCB makes it imperative that confidence in the controls it provides be established. Naturally, this applies to TCB hardware, software, and firmware. The following discussion considers only software verification.[5] Minimizing the complexity of TCB software is a major factor in raising the confidence level that can be assigned to the protection mechanisms it provides. Consequently, two general design goals to follow after identifying all security relevant operations for inclusion in the TCB are (a) to exclude from the TCB software any operations not strictly security-related so that one can focus attention on those that are,[6] and (b) to make as full use as possible of protection

---

[5] Techniques for verifying hardware correctness have tended to emphasize exhaustive testing, and will no doubt continue to do so. Even here, however, the trend is toward more formal techniques of verification, similar to those being applied to software. One approach is given in [Furtek 78]. IBM has done some work on microcode verification.

[6] In order to enhance performance, non-security related software may indeed be placed in the TCB, but this is discouraged.

features available in the hardware. Formal techniques of verification, such as those discussed in the next section, are promoted in TCB design to provide an acceptable methodology upon which to base a decision as to the correctness of the design and of the implementation.

3.3.3.1 Security Model. Any formal methodology for verifying the correctness of a TCB must start with the adoption of a mathematical model of the desired protection policy. A model encompassing mandatory security and to some extent the discretionary security and integrity policies was developed by Bell and LaPadula [Bell and LaPadula 73].[7] There are five axioms of the model. The primary two are the simple security condition and the *-property (read star-property). The simple security condition states that a subject cannot observe an object unless the security level of the subject, that is, the protection attributes, is greater than or equal to that of the object. This axiom alone might be sufficient if not for the threat of non-TCB software either accidently or intentionally copying information into objects at lower security levels. For this reason, the *-property is included. The *-property states a subject may only modify an object if the security level of the subject is less than or equal to the security level of the object.

The simple security condition and the *-property can be circumvented within a computer system by not properly classifying the object initially or by reclassifying the object arbitrarily. To prevent this, the model includes two additional axioms: the activity axiom guarantees that all objects have a well-defined security level known to the TCB; the tranquility axiom requires the classifications of objects are not changed.

The model also defines what is called a "trusted subject" that may be privileged to violate the protection policy in some ways where the policy is too restrictive. For instance, part of the TCB might be a "trusted process" that allows a user to change the security level of information that should be declassified (e.g., has been extracted from a classified document but is itself not classified).

---

[7] Biba has shown how mandatory integrity is the dual of security and, consequently may be modeled similarly.

This action would normally be considered a tranquility or *-property violation, depending on whether the object containing the information had its security level changed or the information was copied into an object at a lower security level.

3.3.3.2 Methodology. A verification methodology is depicted in figure 1.[8] In this technique, the correspondence between the implementation (here shown as the machine code) and protection policy is proven in three steps: (a) the properties of a mathematical model of the protection policy are proven to be upheld in a formal top level specification of the behavior of a given TCB in terms of its input, output, and side effects; (b) the implementation of the specifications in a verifiable programing language[9] is shown to faithfully correspond to the formal specifications; and finally (c) the generated machine code is demonstrated to correctly implement the programs. The model describes the conditions under which the subjects in the system access the objects. With this approach, it can be shown that the machine code realizes the goals of the model, and as a result, that the specified protection is provided.

Where trusted subjects are part of the system, a similar correspondence proof starting with an additional model of the way in which the trusted subject is allowed to violate the general model becomes necessary. Clearly, the more extensive the duties of the trusted subject, the more complex the model and proof.

---

[8]The Hierarchical Development Methodology, developed at SRI International [Robinson et al. 77], is another fairly general methodology for the design, verification, and implementation of reliable software. It has been used for instance to show that software meets certain performance requirements.

[9]The term "verifiable programing language" refers to languages such as Pascal, Gypsy, Modula, and Euclid for which verification tools either exist or are currently being planned. [Good 78]

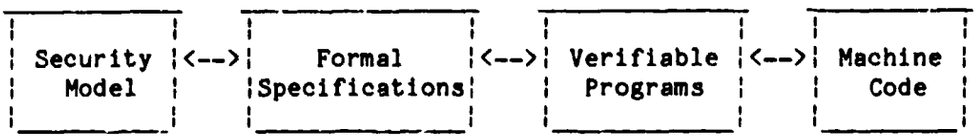

Figure 1. Correspondence Chain

3.3.3.3 Confinement Problems. The TCB is designed to "confine" what a process can access in a computer system. The discussion above centers around direct access to information. Other methods exist to compromise information that are not always as easily detected or corrected. Known as "indirect channels", they exist as a side-effect of resource-sharing. This manner of passing information may be divided into "storage" channels and "timing" channels.[10] Storage channels involve shared control variables that can be influenced by a sender and read by a receiver, for instance when the fact that the system disk is full is returned to a process trying to create a file. Storage channels, however, can be detected using verification techniques. Timing channels also involve the use of resources, but here the exchange medium is time; these channels are not easily detected through verification. An example of a timing channel is where modulation of scheduling time can be used to pass information.

In order to take advantage of indirect channels, at least two "colluding" processes are needed, one with direct access to the information desired, and a second one to detect the modulations and translate them into information that can be used by an unauthorized recipient. Such a channel might be slowed by introducing noise, for instance by varying the length of time certain operations take to complete, but performance would be affected.

Storage channels are related to the visibility of control information: data "about" information, for example, the names of

[10] The terminology in this area in the literature is very confusing. The definitions given here correspond to those used in [Schaefer et al. 77]. See [Lampson 73] and [Lipner 75] for another interpretation.

files not themselves directly accessible, the length of an IPC message to another user, the time an object was last modified, or the access control list of a file.[11] Even the name of a newly created object such as a file can be a channel if this name is dependent on information about other files, e.g., if the name is derived from an incremental counter, used only to generate new file names. This type of channel can often be closed by making the data about legitimate information as protected as the information itself. However, this is not always desirable: for instance, in computer networks, software concerned only with the transmission of messages, not with their contents, might need to view message headers containing message length, destination, etc.

Systems designers should be aware of confinement problems and the threats they pose. Formal techniques to at least identify and determine the bandwidth of the channels, if not completely close them, are certainly of value here. Ad hoc measures may be necessary in their absence.

3.4 PERFORMANCE REQUIREMENTS

Since the functions of the TCB are interpretive in nature, they may be slow to execute unless adequate support is provided in the hardware. For this reason, in the examples of functions given below, hardware implementations (including firmware/microcode), as opposed to software, are stressed, with the idea that reasonable performance is only accomplished when support for the protection mechanisms exists in hardware. Certainly, software implementations are not excluded, and due to the malleability of software, are likely more susceptible to appreciable optimization.

---

[11] It is often the case that even the fact that an object with certain protection attributes exists is information that must be protected.

SECTION 4

DETAILED REQUIREMENTS

The kinds of functions that would be performed by a TCB are outlined below. Those listed are general in nature: they are intended to support both general-purpose operating systems and a variety of dedicated applications that due to potential size and complexity, could not easily be verified.

The functions can be divided into two general areas: software interface functions, operations invoked by programs, and user interface functions, operations invoked directly by users. In terms of a security kernel implementation, the software interface functions would for the most part be implemented by the kernel; the user interface functions would likely be carried out in trusted processes.

4.1 SOFTWARE INTERFACE FUNCTIONS

The TCB acts very much like a primitive operating system. The software interface functions are those system calls that user and application programs running on top of the TCB in processes may directly invoke. The software interface functions fall into three categories: processes, input/output, and storage.

In the descriptions that follow, general input, output, and processing requirements are stated. Output values to processes in particular could cause confinement problems (i.e., serve as indirect channels), by relating the status of control variables that are affected by operations by other processes. Likely instances of this are mentioned wherever possible.

4.1.1 Processes

Processes are the primary active elements in the system, embodying the notion of the subject in the mathematical model. (Processes also behave as objects when communicating with each other.) By definition, a process is "an address space, a point of execution, and a unit of scheduling". More precisely, a process consists of code and data accessible as part of its address space; a program location at which at any point during the life of the process the address of the currently executing instruction can be found; and periodic access to the processor in order to continue. The role of

the TCB is to manage the individual address spaces by providing a unique environment for each process, often called a "per-process virtual space", and to equitably schedule the processor among the processes. Also, since many applications require cooperating processes, an inter-process communication (IPC) mechanism is required as part of the TCB.

4.1.1.1 Create Process. A create process function causes a new per-process virtual space to be established with specific program code and an identified starting execution point. The identity of the user causing the process to be created should be associated with the process, and depending on the protection policy in force, protection attributes should be assigned, such as a security level at which the process should execute in the case of mandatory security.

4.1.1.2 Delete Process. A delete process function causes a process to be purged from the system, and its virtual space freed. The process is no longer considered a valid subject or object. If one process may delete another with different protection attributes, an indirect channel may arise from returning the fact of the success or failure of the operation to the requesting process.

4.1.1.3 Swap Process. A swap process function allows a process to become blocked and consequently enable others to run. A TCB implementation may choose to regularly schedule other processes to execute after some fixed "time-slice" has elapsed for the running process.[12] In order to address a denial of service threat, this will not be the only process blocking operation: certain I/O operations should cause the process initiating the operation to be suspended until the operation completes.

For example, the hardware could support such an operation through mechanisms that effect fast process swaps with the corresponding change in address spaces. An example of such support is a single "descriptor base" register that points to descriptors for a process' address space, only modifiable from the privileged domain. The swap would be executed in little more than the time required for a single "move" operation.

As was mentioned above, the "scheduling" operation in itself may contribute to a timing channel, that must be carefully monitored.

---

[12] If a TCB supports time-slicing, a swap function may not be necessary.

4.1.1.4 IPC Send. A process may send a message to another process permitted to receive messages from it through an IPC send mechanism. The TCB should be guided by the applicable protection policy in determining whether the message should be sent, based on the protection attributes of the sending and receiving process. The TCB should also insure that messages are sent to the correct destination.

An indirect channel may result from returning the success or failure of "queuing" the message to the sending process, because the returned value may indicate the existence of other messages for the destination process, as well as the existence of the destination process. This may be a problem particularly where processes with different protection attributes are involved (even if the attributes are sufficient for actually sending the message). If such a channel is of concern, a better option might be to only return errors involving the message itself (e.g., message too long, bad message format). Clearly, there is a tradeoff here between utility and security.

4.1.1.5 IPC Receive. A process may receive a message previously sent to it through an IPC receive function. The TCB must insure that in allowing a process to receive the message, the process does not violate the applicable protection policy.

4.1.2 Input/Output

Depending on the sophistication of the TCB, I/O operations may range from forcing the user to take care of low level control all the way to hiding from the user all device dependencies, essentially by presenting I/O devices as simple storage objects, such as described below. Where I/O details cannot be entirely hidden from the user, one could classify I/O devices as devices that can only manipulate data objects with a common protection attribute at one time (such as a line printer), and those that can manage data objects representing many different protection attributes simultaneously (such as disk storage devices). These two categories can be even further broken down into devices that can read or write any location in memory and those that can only access specific areas. These categories present special threats, but in all cases the completeness criteria must apply, requiring that the TCB mediate the movement of data from one place to another, that is, from one object to another. To resolve this problem, all I/O operations should be mediated by the TCB.

Some computer architectures only allow software running in the most privileged mode to execute instructions directing I/O. As a result, if only the TCB can assume privileged mode, TCB mediation of I/O is more easily implemented.

In the first category, if access to the device can be controlled merely by restricting access to the memory object which the device uses, the problem becomes how to properly assign the associated memory to a user's process, and no special TCB I/O functions are necessary. However, if special timing requirements must be met to adequately complete an I/O operation, quick response times may only be possible by having the TCB service the device, in which case a special operation is still needed.

When the device can contain objects having different protection attributes, the entire I/O operation will involve not only a memory object, but also a particular object on the device having the requisite protection attributes. TCB mediation in such a case is discussed under "Storage Objects."

4.1.2.1 Access Device. The access device function is a directive to the TCB to perform an I/O operation on a given device with specified data. The operations performed will depend on the device: terminals will require read and write operations at a minimum. The TCB would determine if the protection attributes of the requesting process allow it to reference the device in the manner requested.

This kind of operation will only be necessary when mapped I/O is not possible.

4.1.2.2 Map Device. The map device operation makes the memory and control associated with a device correspond to an area in the process' address space. As in the case of the "access device" function, a process must have protection attributes commensurate to that of the information allowed on the device to successfully execute this operation. This operation may not be possible if mapped I/O is not available in the hardware.

4.1.2.3 Unmap Device. The unmap device frees a device mapped in the address space of a process.

4.1.3 Storage Objects

The term "storage objects" refers to the various logical storage areas into which data is read and written, that is, areas that are recognized as objects by the TCB. Such objects may take the form of logical files or merely recognizable units of a file such as a

fixed-length block. These objects may ultimately reside on a long-term storage device, or only exist during the lifetime of the process, as required. Where long-term devices have information with varied protection attributes, as discussed in the previous section, TCB mediation results in virtualizing the device into recognizable objects each of which may take on different protection attributes. The operations on storage objects include creation, deletion, and the direct access involved in reading and writing.

4.1.3.1 Create Object. The create object function allocates a new storage object. Physical space may or may not be allocated, but if so, the amount of space actually allocated may be a system default value or specified at the time of creation.

As mentioned above, naming conventions for storage objects such as files may open an undesirable indirect channel. If the names are (unambiguously) user-defined or randomly generated by the TCB, the channel can be reduced.

4.1.3.2 Delete Object. The delete object function removes an object from the system and expunges the information and any space associated with it. The TCB first must verify that the protection attributes of the process and object allow the object to be deleted. Indirect channels in this case are similar to those for "delete process". The fact of the success or failure of the operation may cause undesirable information leakage.

4.1.3.3 Fetch Object. The fetch object function makes any data written in the object available to the calling process. The TCB must determine first if the protection attributes of the object allow it to be accessed by the process. This function may be implemented primarily in hardware, by mapping the physical address of the object into a virtual address of the caller, or in software by copying the data in the object into a region of the caller's address space.

4.1.3.4 Store Object. The store object function removes the object from the active environment of the calling process. If the object is mapped into the caller's virtual space, this function will include an unmap.

4.1.3.5 Change Object Protection Attributes. A protection policy may dictate that subjects may change some or all of the protection attributes of objects they can access. Alternatively, only trusted subjects might be allowed to change certain attributes. The TCB should determine if such a change is permitted within the limits of the protection policy.

4.2 USER INTEPFACE FUNCTIONS

The TCB software interface functions address the operations executable by arbitrary user or applications software. The user interface functions, on the other hand, include those operations that should be directly invokable by users. By localizing the security-critical functions in a TCB for verification, it becomes unnecessary for the remaining software running in the system to be verified before the system can be trusted to enforce a protection policy. Most applications software should be able to run securely, by merely taking advantage of TCB software interface facilities.[13] When users need capabilities beyond that normally provided to general applications, such as the ability to change the owner of a file object, direct contact with the TCB is required.

In kernel-based systems, the user interface functions are commonly implemented as trusted processes. Moreover, these trusted processes rely on the equivalent of the software interface functions for support.

These functions fall into three categories: user services, operations and maintenance, and administration.

4.2.1 User Services

Certain operations may be available to users as part of standard set of functions a user may wish to perform. Three are of interest here: authentication of the user to the system and of the system to the user, modification of protection attributes, and special I/O.

4.2.1.1 Authentication. The act of "logging in", of identifying oneself to the system and confirming that the system is ready to act on the behalf of the requestor, is critical to the protection mechanisms, since all operations and data accesses that subsequently occur will be done in the name of this user. Consequently,

---

[13] Applications may enforce their own protection requirements in addition to those of the TCB, e.g., a data base management system may require very small files be controlled, where the granularity of the files is too small to be feasibly protected by the TCB. In such a case, the application would still rely on the basic protection environment provided by the TCB.

identification and authentication mechanisms that play a part in validating a user to the system should be carefully designed and implemented as part of the TCB.

Likewise, the system must have some way of alerting the user when the TCB is in command of terminal communications, rather than untrusted software merely mimicking the TCB. For example, the TCB might signal to the user in a way that non-TCB software could not, or a special terminal button could be reserved for users to force the attention of the TCB, to the exclusion of all other processes.

4.2.1.2 Access Modification. Access modification functions allow a user to securely redefine the protection attributes of objects he/she controls, particularly in the case of discretionary policy. Also included here are operations that allow a user to select the protection attributes to be assumed while using the system, where the attributes may take on a range of values. For example, a user with a security level of Top Secret, may choose temporarily to operate as if Unclassified in order to update bowling scores.

Many factors must be considered in implementing such an operation, particularly if implemented in a process. The user must have some way of convincing himself that the object for which the protection attributes are being changed is indeed what is intended. For instance, the user might be allowed to view a file to confirm its contents before changing its security level. Another issue involves the synchronization problem resulting from other processes possibly accessing the object at the instant the access modification is attempted. The TCB should prevent such a change from occurring unless the object were "locked", or temporarily made inaccessible to other processes, until the operation was complete, and also access to the other processes should be re-evaluated on completion.

4.2.1.3 Special I/O. I/O functions not covered in the software interface functions due to their specialized nature are: (a) network communications, and (b) spooling, e.g. to a line printer or mailer. The ramifications of both of these areas are too extensive to adequately cover here. The reader is referred to [KSOS 78].

4.2.2 Operations/Maintenance

In the operations and maintenance category fall those functions that would normally be performed by special users, the system operators, in running and maintaining the system. Examples of such operations are system startup and shutdown, backup and restore of long-term storage, system-wide diagnostics, and system generation.

4.2.2.1 Startup/Shutdown. The security model discussed above assumes that in a TCB, an initial secure state is attained and that subsequent operations on the system obey the protection policy and do not affect the security of the system. This characteristic of a TCB can be said to be true regardless of the protection policy and security model employed. A "startup", or bootstrap, operation addresses the initialization of the system and the establishment of the protection environment upon which subsequent operations are based. The model itself, or the formal specifications of a specific design, can address what the characteristics of all secure states are, and hence the requirements for the initial secure state. Consequently, programs that create this state can be well-defined. Since it is the operator who must execute the necessary procedures that initialize the system, TCB functions interfacing the operator must be trusted to do what the operator specifies.

Shutdown procedures are equally crucial in that an arbitrary suspension of system activities could easily leave the system in an incomplete state, making it difficult to resume securely (for instance, if only half of an updated password file is moved back to disk). One must, for instance, write all memory-resident tables out to disk where necessary.

4.2.2.2 Backup/Restore. To allow for recovery from unpredictable hardware failure, and consequently the arbitrary suspension mentioned above, "checkpoints" may be taken of a given state of the storage system, for instance, by copying all files from disk to some other medium, such as magnetic tape. In the event of system failure, the state of files at some earlier time can be recovered. The backup function must operate on the system in a consistent state, and accurately reflect that state; the restore function must reliably rebuild from the last completely consistent record it has of a secure state. Note that the backup system requires an especially high level of trust since it stores protection attributes as well as data.

4.2.2.3 Diagnostics. Diagnostics of both hardware and software integrity can thwart potentially harmful situations. In particular, hardware diagnostics attempt to signal when problems arise, or, when something has already gone wrong, they try to aid the technician in pinpointing where the problem is. Diagnostics written in software typically access all areas of memory and devices, and consequently, if run during normal operation of the rest of the system, require tight TCB controls. If possible, they should be relegated to user programs and limited to specific access spaces during the course of their operation. However, in such a case it would be impossible to

test the security critical hardware, such as descriptor registers if present. Such software, for on-line diagnosis, must be included in the TCB, and limited to operator use.

4.2.2.4 System Generation. System generation deals with creating the program modules in executable form that can subsequently be loaded during system startup. It is included here for completeness, although there is no intention in this report to require that editors, compilers, loaders, and so forth, be verified to correctly produce the code that is later verified correct. Correct system generation is an area that is clearly vulnerable, and procedures must be made to ensure that the master source is not intentionally corrupted.

4.2.3 Administration

The administration and overall management of a system both in terms of daily operations and security operations may be relegated to a user, or users, other than the system operator. Functions in support of system administration include but are not limited to updating data bases of users and their valid protection attributes; and audit and surveillance of protection violations

4.2.3.1 User Data Base Updates. A typical user data base would contain at a minimum the names of valid users, their authentication data (e.g., password, voice print, fingerprints), and information relating to the protection attributes each user may take on while using the system. TCB functions must be available to an administrator to allow updates to the data base in such a way that the new information is faithfully represented to the user authentication mechanism.

4.2.3.2 Audit and Surveillance. Audit facilities capture and securely record significant events in the system, including potential protection violations, and provide functions to access and review the data. Surveillance facilities allow for real-time inspection of system activities. Audit and surveillance mechanisms provide an additional layer of protection. They should be implemented as part of a TCB not only because they require access to all activities on the system as they occur, but also since if they are not themselves verified to be correct and complete, flagrant violations might go undetected.