75

MITRE-Bedford Division

MTR-3267
Vol. I

Multics Security Evaluation: Results and Recommendations

S. B. Lipner

OCTOBER 1978

MITRE

[illegible]

76

MITRE-Bedford Division

MTR-3267
Vol. I

Multics Security Evaluation: Results and Recommendations

S. B. Lipner

OCTOBER 1978

MITRE

MITRE Technical Report
MTR-3267
Vol. I

Multics Security Evaluation: Results and Recommendations

S. B. Lipner

OCTOBER 1978

CONTRACT SPONSOR	ESD
CONTRACT NO.	F19628-79-C-0001
PROJECT NO.	5720
DEPT.	D-75

THE MITRE CORPORATION
BEDFORD, MASSACHUSETTS

Department Approval: [illegible]
Edward H. Bensley

MITRE Project Approval: [illegible]
Edward H. Bensley

ABSTRACT

The Honeywell Multics computer system has been proposed for possible use in a multilevel secuity environment at the Air Force Data Service Center. This report presents an overview of the security aspects of the Multics operating system design, and the results of an ESD/MITRE security evaluation of Multics for the obsolete Honeywell 645 processor. The results of the 645 penetration and design review are used to project the probable security strengths and weaknesses of the current Honeywell 6180 Multics. While the use of Multics in an open environment is not recommended, a set of actions that can support the use of Multics in a controlled multilevel environment is identified and recommended.

ACKNOWLEDGEMENT

The author acknowledges the contributions of Lt. Col. Roger R. Schell and Capt. Paul A. Karger, both (then) of ESD/TOIT, during the conduct of the security evaluation reported here. Both Lt. Col. Schell and Capt. Karger also commented on the drafts of this report.

A number of other personnel from ESD and MITRE have also read drafts of this report and offered their suggestions. The contributions of these people are gratefully acknowledged.

Mauro Ferdman of MITRE assumed the responsibility for final editing and production of the report. His contribution merits special thanks.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

SECTION 1

INTRODUCTION

This report presents the results of an ESD/MITRE assessment of the Honeywell Multics computing system as a candidate for use in a controlled multilevel security environment at the Air Force Data Services Center (AFDSC). The study was conducted in the period from July 1972 to October 1973. The major period of activity was October 1972 to March 1973.

Other volumes published separately present specific results of examining in detail the security and utility of Multics; this report attempts to tie those results together and present overall conclusions regarding AFDSC's possible use of Multics. Because the major motivation for examining Multics was its potential for secure computing, most of this report deals with the current state and future prospects of computer security in the Multics environment. The remaining paragraphs of this section outline the background of the ESD/MITRE study, summarize key findings and conclusions, and present an overview of the remainder of the report.

BACKGROUND

The Air Force Data Services Center operates a general-purpose computer utility in the Pentagon, serving a number of Air Force and DoD agencies with several computers, principally two dual-processor Honeywell 635 systems. The diversity of user requirements for classified and unclassified processing, and the diversity of user locations requiring on-line access led AFDSC in 1970 to request that Air Force Systems Command develop a method of operating the 635 computers in a multilevel secure mode. The requirement for secure operation at AFDSC was expressed by Development Directive 79 from Headquarters USAF and pursued under AFSC Project 6917.

An ESD/MITRE study conducted under Project 6917 in 1970 determined that providing a multilevel security mode would require, at minimum, a complete redesign and reimplementation of the GCOS operating system for the 635. More important, the study held out little hope for the achievement of security in any operating system for the 635, no matter how extensively modified. Subsequent examinations of 635 (and follow-on HIS 6000 and Level 66) "secure operating systems" have tended to confirm this judgement. Thus the AFDSC requirement for secure multilevel operation expressed by Development Directive 79 remained unsatisfied.

In 1972, the Office of the Assistant Secretary of Defense (Systems Analysis) conducted a study of alternative computer systems that might provide it with needed time-sharing services. AFDSC suggested to OSD(SA) that an AFDSC-operated Multics system might meet the OSD(SA) requirement. However, a problem was again presented by security: Much of the OSD(SA) workload is classified top secret while most AFDSC users are cleared and most AFDSC work is classified only at the secret level. The Multics system is too large and costly to be dedicated to the OSD(SA) workload; thus some way must be found to serve the secret-cleared AFDSC users as well as the top secret OSD(SA) demand on the same machine. An ESD preliminary study suggested that Multics might provide the required security for a controlled environment of secret and top secret-cleared users. Based on the preliminary study and other considerations, AFDSC was directed to acquire the Multics system and operate it in support of OSD(SA) as well as its own users.

This ESD/MITRE assessment of Multics is intended to provide AFDSC with information on the advisability and limitations of a two-level Multics system in a secret-top secret controlled environment. At the same time, the study considers the possible future use of Multics in a more general multilevel environment to satisfy AFDSC requirements. As such it attempts to point the way toward satisfying the total need expressed in Development Directive 79.

Preliminary results of this study indicated that a Multics computer could be used in a controlled multilevel environment. On the basis of these results, a Request for Proposal (RFP) to Honeywell was prepared and issued in late 1972 and early 1973. Numerous results of this study are reflected in the RFP, especially in those portions dealing with required security measures.

SUMMARY OF FINDINGS

This study effort has included an exhaustive examination of available Multics documentation--primarily that available in the technical literature plus user-level manuals. A more limited examination addressed the system programs and system documentation of the existing (HIS 645) Multics computer. Several penetration efforts were directed at the 645 Multics. Only very limited attention has been directed at the follow-on hardware (HIS 6180 and Level 68) and software that would actually be installed at AFDSC, for neither was available during the study period.

Beardsley commented in [1] that "Graduate students have always been the best test of system security...Multics has, in fact, probably been the object of more penetration attempts than all other time-sharing facilities combined". However, this report supports the alternative point of view that a deliberate, planned penetration attempt is a far more severe test. The efforts at penetrating 645 Multics given ordinary user's access were completely successful. Given this fact, it is unwise to operate a 645 Multics system or its immediate successor Level 68 Multics in an "open" multilevel security environment where it is subject to programmed attacks by potentially hostile users. On the other hand, operation in a controlled multilevel environment with users cleared for either secret or top secret information and user areas protected to top secret is an acceptable option. In the latter controlled environment, the system hardware and software are protecting mainly against accidental disclosures and, with modifications outlined below in the areas of explicit classification and input-output, are quite adequate.

In a somewhat longer time frame, Multics offers the basis for a system that can provide effective security in an open environment. The HIS Level 68 hardware includes those features [2] that appear to be necessary to support a provably correct "security kernel" [3] that is capable of assuring protection of information from unauthorized access or modification. The basic Multics software concept provides for a uniform virtual machine environment and for complete checking of programs' attempts to access information. Both features provide significant aids to system security. Thus it is reasonable to expect that a secure system based on Multics could be achieved by restructuring in a known way rather than a never-ending search for bugs and errors. Such a system would be of significant utility to AFDSC and throughout DoD, for a Multics computer can support many existing programs written for the HIS 635 and 6000 series processors and the GCOS operating system. A secure system based on Multics hardware and a security "kernel" would have a probability of security failure corresponding to the (low) probability of failure of a security related hardware component. In contrast, security failures in systems with software security flaws, such as the HIS 635, can be produced repeatedly by a hostile agent.

A brief examination of the utility of Multics revealed no unacceptable penalties for system use or security. The Multics design emphasized overall user service rather than maximum batch throughput. Still, the 645 Multics system compares reasonably in throughput with other contemporary computers; the Level 68 Multics processor has greater basic speed and several new hardware features, so its performance should be quite competitive. The use of hardware for most security checks and the provision of a segmented virtual memory reduce the performance penalties for security to an insignificant level [4].

The development objectives for Multics included the provision of a flexible environment in which user-oriented software packages could be built, but did not call for the provision of such packages as part of the basic system. An examination of the human interface manifest by the Multics commands, editors, computer-aided instruction package, and manuals shows the present system to have been written "by and for programmers". While Multics presents the non-programmer user a face no worse (except for documentation) than those of other contemporary systems, it is not any better. However Multics does meet its objective and have the potential to provide a significantly improved human interface, given thought, commitment, and effort.

OVERVIEW OF THE REPORT

The following six sections present a complete picture of the ESD/MITRE evaluation of Multics, especially in the realm of security. Details of specific evaluations, including the penetrations of 645 Multics, are reported in separate volumes.

Section 2 below describes the scope of consideration of Multics--what aspects of the system were examined and to what level of detail. Section 3 presents a summary of the Multics access control design. Section 4 describes ESD/MITRE experiences in evaluating the security of the existing 645 Multics. Section 5, somewhat speculative in nature, considers the prospects for security in an initial 6180 Multics delivered to AFDSC. Section 6, based in part on ongoing ESD/MITRE development efforts, discusses the mid-term prospects for a secure system based on Level 68 or 6180 Multics. The final section presents the recommendations of the ESD/MITRE evaluation of Multics.

SECTION 2

SCOPE OF THE EVALUATION

INTRODUCTION

This section describes the scope of consideration of Multics by the ESD/MITRE evaluation group--what aspects of the system were examined, to what depth, and from what sources. During the period of the study, Multics evolved from a prototype system operated at four sites on hand-modified second generation hardware to an announced Honeywell product supported by third-generation hardware. As the study was completed, the evolution was not complete, and the depth of documentation of and experience with the Multics delivered to AFDSC was rather uneven. The following paragraphs describe the scope of consideration of system design, system implementation (the vulnerability analysis or penetration), development prospects, and system utility.

SECURITY SYSTEM DESIGN

The basic design of the Multics hardware-software system and its access controls is accurately described in a number of papers published in technical journals and proceedings between 1965 and the present. These papers formed the basis for the evaluation of the Multics security and access control design. At a more detailed level, the security system design is reflected in the system descriptions included in the _Multics Programmer's Manual_. An examination of papers and manuals led to the conclusion that input-output to demountable storage media and handling the military system of classifications and special access categories would be special problems in an initial secure Multics system. Accordingly, special attention was paid to these two areas. A related document [5] discusses the problems of demountable media input-output, while considerations of classification and special-access category form a major part of the security system specifications included in the RFP issued to Honeywell.

SECURITY SYSTEM IMPLEMENTATION

The basic reference on the implementation of the Multics security controls is the Multics system itself--the source listings of the operating system modules, the _Multics System Programmer's Supplement_ and the hardware manual describing the HIS 645 processor.

Given these references, the ESD/MITRE group charged with evaluating the security of Multics identified a number of potential vulnerabilities and constructed penetration programs to exploit them. While the identification of vulnerabilities was aided by the presence on the evaluation team of two individuals familiar with Multics, _their presence made the penetrations possible with extremely low effort, rather than making the difference between success and failure of the penetration efforts_. The familiarity with Multics required to effect penetration was no greater than that required of any system programmer working in an installation that uses Multics. Specific points of attack included the HIS 645 hardware, the operating system software, the maintenance procedures and security "features". The level of effort expended on the penetrations (about two man-months) was low in comparison with that expended by other contemporary "penetration teams".

DEVELOPMENT PROSPECTS

During 1972, ESD sponsored a Computer Security Technology Planning Study Panel [3] composed of experts in secure computing drawn from government, industry, and the academic community. The efforts of this panel, and subsequent ESD-sponsored efforts to expand on and exploit its recommendations, have led to the conclusion that the HIS Level 68 offers a sound basis for the development of a secure computer system. While that conclusion is a result of a major effort, the effort is aimed at developing a prototype secure computer system, rather than at evaluating Multics for AFDSC.

SYSTEM UTILITY

The assessment of system performance is built on comparative performance data assembled by a previous MITRE project. This project involved executing the scientific benchmark job (Job 14) of the WWMCCS benchmark test series on a number of computers including the IBM 370/155 and 360/50, and the HIS 635, 6050 and 6070. For the Multics performance assessment, the benchmark program was converted to Multics Fortran for execution on the HIS 645 and 6180. The results of the performance assessment are fully documented in a separate volume [6].

The evaluation of the Multics human interface was focused on an examination of the TICS computer-aided instruction package proposed as a training tool for AFDSC security and user personnel. The examination of TICS involved developing a small tutorial to teach secretaries the use of the qedx editor as a tool for document preparation. Sufficient experience was gained to support the observations that some useful Multics commands and subsystems were written more for their programmer authors than for outsiders, and that the interface the commands and subsystems present to a user is not always consistent, mnemonic, or understandable [7]. The documentation in many cases aggravates this problem by being more of a specification or reference manual than a user's guide. Honeywell is aware of this problem and is pursuing an effort to upgrade the Multics user documentation in conjunction with their efforts to make the Level 68 a commercial product.

SECTION 3

THE MULTICS ACCESS CONTROL DESIGN

INTRODUCTION

This section describes the design of Multics with emphasis on the access controls--those parts of the system that control what a user (or his program) may read, write, or execute. The purpose of this section is to give the reader sufficient background to understand why the basic design of Multics is considered sound from an access control standpoint, and what areas of the design include points of potential risk or require future development. The description is not intended to be complete or definitive, for numerous documents in the literature [8] [9] [10] [11] [12] detail the design of Multics.

The first and largest of the subsections below describes the Multics virtual memory--the major element of the access control environment. A second shorter subsection discusses the access control implications of the Multics input/output system. A brief conclusion summarizes the security strengths and weaknesses of the Multics design.

THE MULTICS VIRTUAL MEMORY

The major unique feature of the Multics design is its provision to the programmer of a segmented virtual memory. This subsection discusses in turn the basic properties of the segmented virtual memory, the interface between user and supervisor programs (or more generally between programs of lesser and greater privilege), and the underlying memory management design (paging) that supports the virtual memory.

Segmentation

Segments, Directories and the File System

Each user active on Multics is normally associated with his own "virtual processor" that executes programs and accesses information on his behalf. A conventional processor reads or writes data and executes instructions only in some (real or virtual) primary memory; the data or programs permanently or temporarily held in secondary storage files must be explicitly read to primary memory on request of the user's program before the processor can operate on them (Figure 1a). In contrast, the Multics virtual processor operates directly on information in the file system (Figure 1b). The files--referred to as segments--being processed retain their identity instead of being moved into a single amorphous real or virtual "core". This distinction between the Multics and conventional processors lies at the heart of the security implications of the Multics design.

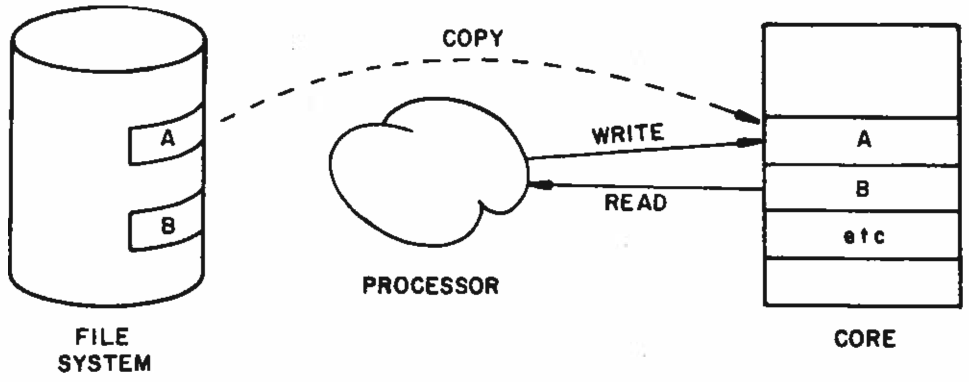
The Multics file system is organized logically as a single tree of segments. Data and programs may only appear in segments that are leaves (at the bottom) of the tree. Other segments are used as directories and contain information about the names, physical locations, and access restrictions of segments lower in the tree. To designate a specific segment, a Multics user or process names, with the aid of conventions and default values, every directory in order from the root of the tree to the desired segment. For example, segment "slow" in figure 2 is designated by:

```
>udd>Druid>Mi>sbl>gored-ox>slow.
```

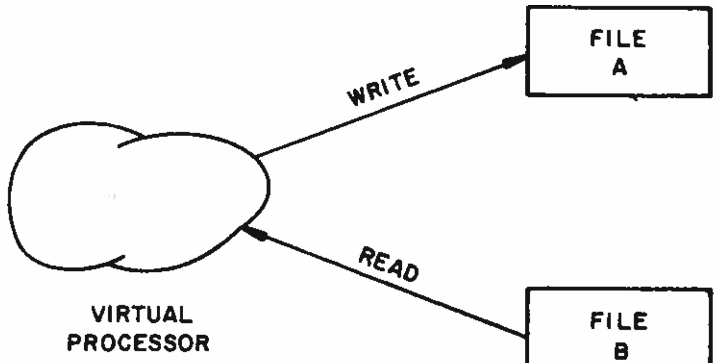
(The name Root is implied before the first >.)

Potentially (subject to access restrictions) any segment in the file system may appear in the address space of any user's virtual processor. The operation of the Multics segment management software requires that if a given segment is to appear in a virtual processor's address space, all directory segments superior to it in the direct path to the root of the tree must also appear in the address space (Figure 2). Both security and correctness of operation require that users' programs be prevented from operating directly on directory segments. This restriction is enforced by the mechanisms (discussed in the next subsection) that distinguish user programs and data from those of the supervisor in a virtual processor's address space. Users or their programs appeal to the supervisor to request operations that involve access or modifications to directories.

A given directory segment (such as gored-ox in figure 2) contains one directory entry or branch for each segment directly beneath it in the tree. (Thus gored-ox would contain four branches--for elephant1, elephant2, slow, and better). Each branch identifies every user who may access the corresponding (data, program, or directory) segment, and the types of access that are allowed for each user. The access types for program or data segments are read, write, and execute, and each has the usual meaning. If a branch specifies a segment that is a directory lower



(a) CONVENTIONAL ORGANIZATION



(b) MULTICS SEGMENTED ORGANIZATION

IA-48,532

Figure I STORAGE ORGANIZATION

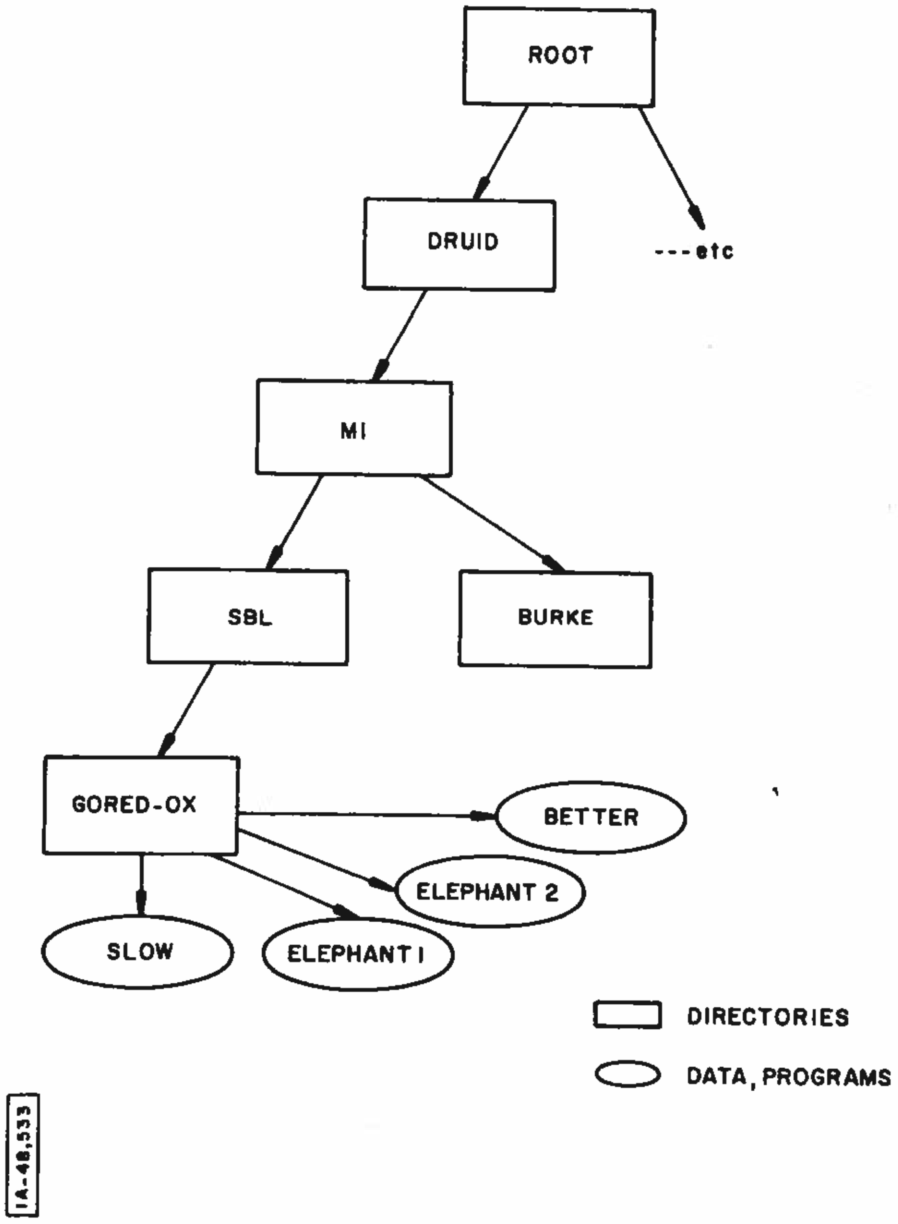


Figure 2 MULTICS FILE SYSTEM HIERARCHY

in the file system tree, the access types in the branch define users' rights to perform operations on the branches of the lower directory. The access types for directory segments are status, allowing a user to list the segment names, sizes and access rights (etc.) specified by a directory's branches; modify, allowing a user to change the branches in a directory so as to change access rights or delete segments subordinate to the directory; and append, allowing a user to add new branches (and subordinate segments) to a directory. Different users may have different access rights to a given segment--one user might be able to read or write a data segment and a second only to read it. Similarly, one user might have status and modify access to a directory segment, a second might have only status access, and a third might have no access at all.

The operation of the directory tree and segment access controls may be illustrated by an example using figure 2. Suppose that a user, say Burke, has no access at all to directories sbl or gored-ox, or to programs slow or better, but does have modify access to directory Mi. Then if he wishes to access slow and better, Burke can alter the branch in Mi that specifies directory sbl, and give himself modify access to sbl. Similarly, he can then modify the branch in sbl that specifies gored-ox and give himself modify access to gored-ox. With this access, he can give himself execute access to slow and better by modifying their branches in gored-ox. Thus a user with modify access to a high-level directory can force access to any lower segment if he wishes. (For this reason, the granting of modify access to high-level directories in Multics must be subject to administrative controls.)

The discussion of access authorizations above stated that each branch in a directory identifies every user who may access the segment corresponding to the branch and the modes of access that each user is allowed. The access control list in the branch and the identification of Multics users are organized to provide administrative simplicity and to prevent the overhead of very long lists of authorized users. Each Multics user is assigned to one or more projects for administrative purposes. When a user logs in, his virtual processor is identified by his name, the project under which he logged in, and a one-letter (process) identifier. Access control lists then specify combinations of user, project, and process identities. Thus user Lipner might be able to access directory Mi in status, modify, or append modes, if he is logged in under project Druid with process identity a (Lipner.Druid.a). The conventions for access control list entries include "don't care" specifications that allow access by a class of users. For example, any user on project Druid might have status access to Mi with any process (*.Druid.*). The order in which access control lists are checked is intended to insure that users will be given the proper access. In the examples above, Lipner.Druid.a will be found before *.Druid.*, and Lipner's virtual processor will be given full access (status, modify, append) to Mi.

Virtual Processor Operations

The initial paragraphs of this subsection discussed the direct addressing of segments in the file system by the Multics virtual processors. Subsequently the organization of the file system and its access controls were described. The remaining paragraphs attempt to establish the relationship among the virtual processors, hardware processors, and file system. The concluding paragraph of this subsection discusses the security implications of the Multics segmented memory.

The Multics processor includes hardware that supports the Multics segmented memory. Every address specified by a Multics program is a "two-dimensional" address that specifies a segment and a word within the segment. While the discussion above considered named segments within the file system, the processor hardware can only interpret segment numbers. A segment number specifies a displacement in a table (the descriptor segment) that contains one descriptor for each segment in the processor's address space. The descriptor for a given segment contains much the same information as the directory branch for that segment: location, size, and access rights. When a program operating on a Multics processor specifies an address (a segment number and word number) the processor uses the segment number as an index to the descriptor segment and finds the segment's location and size, and the access rights of the virtual processor in control of the hardware processor. If the word specified is within the segment (determined by its size) and the access requested is consistent with the virtual processor's access rights, the hardware allows the word to be read, written, or executed as an instruction. If one of the conditions mentioned is not met, the processor refuses the access and immediately transfers (faults) to an appropriate supervisor routine.

As its name implies, the descriptor segment is itself a Multics segment, but one that can only be operated on by a virtual processor that is executing a supervisor routine. Each user's virtual processor has its own descriptor segment. The descriptor segment for the virtual processor in control of the hardware processor at a given time is addressed by a hardware register called the descriptor base register or DBR (figure 3). Thus multiprogramming (switching control of the hardware processor from one virtual processor to another) involves only reloading the DBR. The use of a separate descriptor segment for each virtual processor provides the virtual

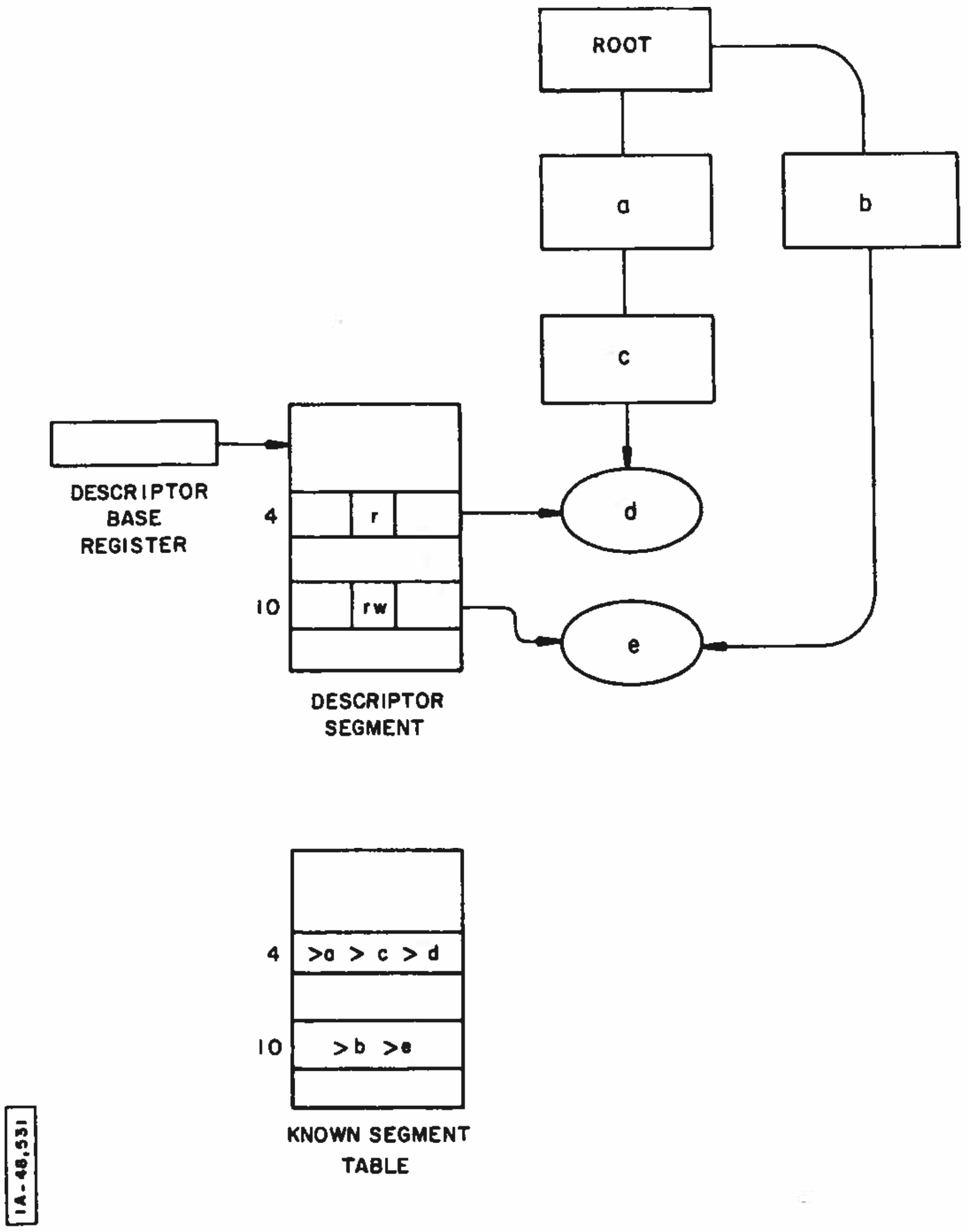


Figure 3 DESCRIPTION FOR SEGMENT AND KNOWN SEGMENT TABLE

processor with access to only the information that would be available to its user if he were executing his program on a dedicated machine.

The descriptor for a segment contains information that is available in the directory branch that identifies the segment. Making a segment directly addressable to a virtual processor, then, requires copying information from the branch for the segment into a descriptor. This operation is called _making a segment known_ to a _process_ (virtual processor in execution) or simply _making known_. When information is copied from the branch to the descriptor, certain additional information is placed in a _known segment table_ (KST) entry corresponding to the descriptor. While information in the descriptor allows the hardware processor to find the segment given its number, information in the KST allows a program running on the virtual processor to find the branch information for the segment (such as its name) given the segment number. The KST entry, like the descriptor, is created by the Multics supervisor when a segment must be accessed by the virtual processor for the first time.

An important aspect of Multics, and one consistent with the discussion of direct addressing of files, is the fact that segment numbers are only a shorthand provided for the convenience of the hardware processor. Commands and programs in Multics use the names of segments in the file system. A set of system linkage conventions provides for the automatic making known of a segment, and the arbitrary assignment of a segment number when the segment is first accessed by a given virtual processor. The program segments created by the Multics language processors follow these conventions, so many Multics users and programmers need never be aware of the mapping from segment name to segment number. The assignment of segment number to segment is completely arbitrary: a given segment may be addressed by a user's virtual processor with one number on one day, and with a different number when the user logs in again (and thus has a "new" virtual processor) the next day. Similarly, two users' virtual processors that are sharing the Multics system by multiprogramming can address the same segment at the same time using two different segment numbers (Figure 4).

The management of shared segments in Multics deserves explicit mention at this point. In a more conventional computer system, if two multiprogrammed processes require simultaneous access to a file, each reads it into "core" (real or virtual) and operates on it as necessary. In Multics, each process is given a descriptor for the segment and accesses it "in place" in the file system. This concept

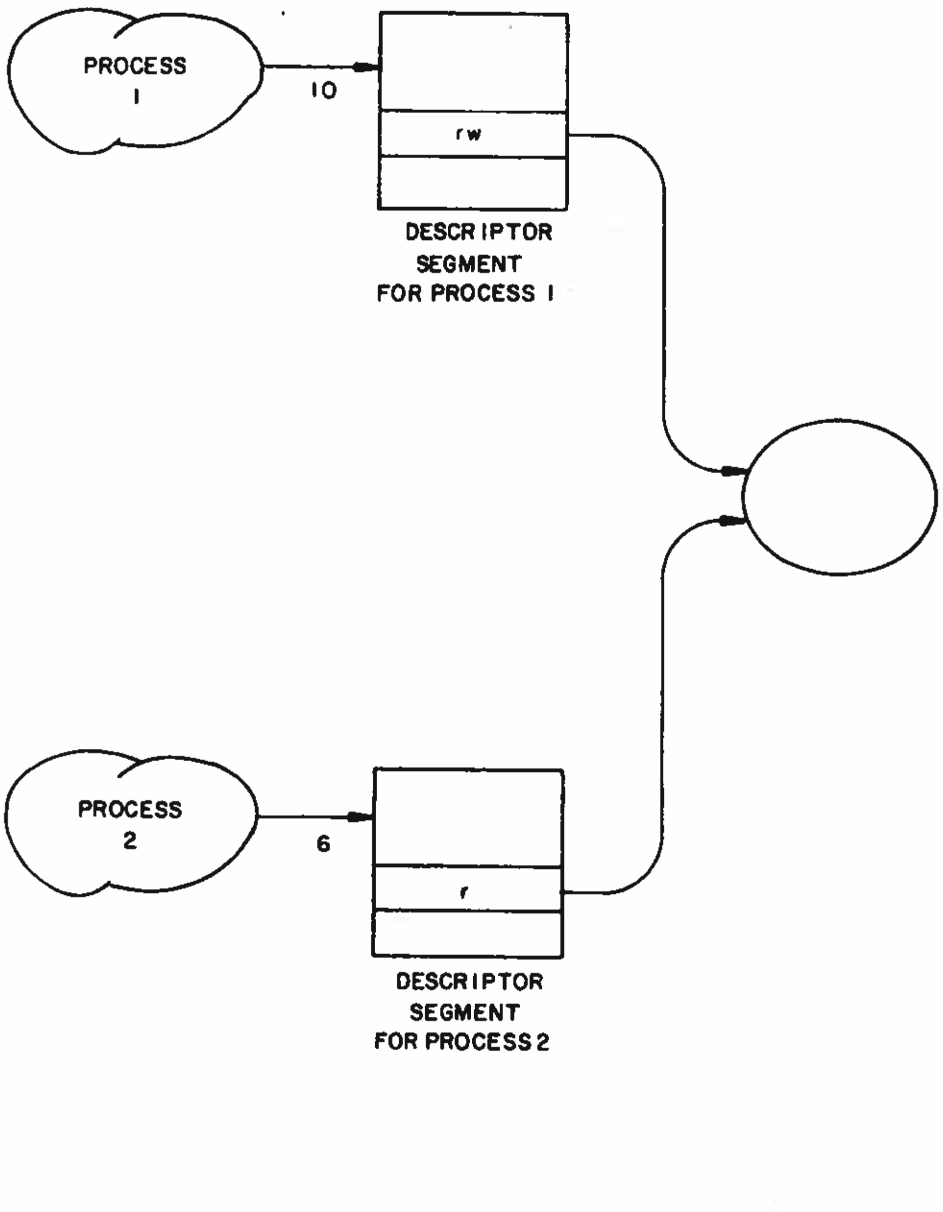

PROCESS 1
10
rw
DESCRIPTOR SEGMENT FOR PROCESS 1
PROCESS 2
6
r
DESCRIPTOR SEGMENT FOR PROCESS 2
IA-48,534

Figure 4   SEGMENT SHARED BY MULTIPLE PROCESSES

of sharing results in significant efficiency in the use of oft-shared programs and data bases.[1] Since each process has a separate descriptor, it may have distinct access rights to the segment. Thus, in figure 4, process 1 might be able to read and write its segment n, while process 2 might only be able to read its segment m, even though both numbers refer to the same segment.

The discussions above have addressed the operation of virtual processors without regard to the distinction between user and supervisor functions. While there are numerous operations that can be performed by the supervisor but not by a user's program, the supervisor, like a user program, operates as part of the user's process and within the segmented environment. Thus when a user program requires a supervisor service, it invokes a supervisor routine. The latter may have more power than a user program (for example, to write the descriptor segment) but is still bound by the user's access rights to segments accessible to the virtual processor. For example, if the user is denied access to segment slow (Figure 2), then the user's virtual processor has no descriptor for slow, and the supervisor, like a user program, has no way to address slow. There is simply no way for the virtual processor to express the address of slow. If the virtual processor wishes to access slow, it must appeal to the "make known" routine in the supervisor. This routine, on discovering that the user has no access in the access control list for slow, will deny the request whether it comes from a user or supervisor program. This situation is in contrast to those of more conventional systems in which the supervisor has access to the computer's entire address space, and can potentially be tricked into addressing any information for a user.

Summary of Segmentation

In summary, the use of segmentation in Multics has the effects of:

(1) Reducing and controlling the power of the supervisor (the "principle of least privilege");
(2) Enforcing explicit access controls to each file in the system for each user; and

----

[1] Of course, sharing does not eliminate the need for processes to avoid inconsistency if they attempt to update the same data base. Multics provides the usual locking mechanisms which may be used as agreed by processes wishing to achieve harmonious cooperation.

(3) Providing a uniform environment for both user and supervisor routines that encompasses the normal functions of primary and secondary storage management systems.

These three effects result in a relatively simple and uniform access control environment for Multics, and one in which security-related supervisor programs can be identified and reviewed.

Protection Rings

The paragraphs above made frequent reference to the mechanisms that distinguish between user and supervisor procedures, or more generally, between procedures of lesser and greater privilege being executed by a virtual processor. These mechanisms in Multics are referred to by the name protection rings, a name drawn from a graphical representation that has been used to describe their operation. The following subsections present an overview of the Multics ring mechanisms, then describe their operations with respect to data segments, procedure segments, and directory segments.[2]

Overview of Rings

At any instant, the processor on which a Multics process is executing has associated with it a domain or ring number. Before it accesses any word of any segment on the process' behalf, the Multics processor compares its ring number with a set of three ring bracket numbers in the descriptor for the segment. The relationship among ring number, ring brackets, access modes recorded in the descriptor, and access mode requested determines whether the access will be allowed or refused. Routines with different levels of privilege (for example, supervisor and user routines) execute within the process, but with ring number and ring brackets set so that information receives appropriate protection. To take an example from the discussions above, a process may write in its own descriptor segment, but only when the processor ring number is set to zero. As a process may only obtain control of the processor "in ring zero" (the most privileged ring) by entering a supervisor routine in a controlled way, user programs are prevented from altering the descriptor segment.

The exact interpretation of the ring brackets for a segment depends, as mentioned above, on whether the segment contains data,

----

[2] A complete description of the Multics protection ring mechanism is beyond the scope of this report. See [11].

procedure code, or a directory. The following subsections discuss each of these cases in turn.

Rings and Data Segments

For a data segment, the two ring brackets of interest are the read bracket and the write bracket. If a process is allowed read access to a segment, it may effect that access while executing in rings numbered from zero to the number specified in the segment's read bracket. Thus the read bracket for a sensitive supervisor data base might be zero (accessible only to the Multics operating system) while that for a globally useful data base would be seven (the highest ring number).

The write bracket for a segment identifies the highest ring in which a process may be executing if it is successfully to attempt to write in the segment. In order for a processor to write in a segment, the processor ring number must lie between zero and the segment's write bracket and the process' descriptor for the segment must indicate that write is an allowed access mode. The provision of separate read and write brackets and access mode bits in the descriptor allows flexibility in the granting of access to a segment. A process might be allowed to read a segment in rings zero through seven and to write it only in (supervisor) rings zero and one. For this case, the read and write access bits would be set, the read bracket would be seven and the write bracket one. Since the descriptors by which segments are accessed are "per-process" (although the ring bracket values are not), a second process might be granted only read access in rings zero through seven for the same segment. For this process, only the read access bit would be set, while the read and write brackets would be seven and one as before.

Rings and Procedure Segments

The ring brackets for a procedure segment govern the setting of the processor ring number while the procedure is in execution. In addition, the procedure ring brackets determine whether the procedure can be invoked while the processor is executing in a given ring. Three restrictions regarding a procedure segment are imposed by the hardware ring mechanisms:

(1) The highest ring in which a procedure can execute (be the target of a successful call or branch instruction) is limited so that a non-privileged routine cannot simply branch to a sensitive one at an arbitrary point;

(2) The lowest numbered ring in which a procedure may execute is limited so that a non-privileged routine may not obtain

control of the processor in a privileged (low-numbered) ring; and

(3) The ability of any routine in a higher-numbered ring to invoke in a controlled way a routine that executes in a lower-numbered ring is provided so that (for example) a user program can request specific services of a supervisor program.

All three ring brackets in the descriptor are used to implement the restrictions outlined above for any segment whose execute access bit is on. The read bracket serves as a _read-execute bracket_ and specifies the highest ring in which a procedure segment may execute. The write bracket specifies both the lowest ring in which a segment may execute and, if the write access bit in the descriptor is on, the highest ring in which a segment may be written. (This mechanism assures that there is only a single ring in which an impure procedure segment may be both written and executed. Were this restriction not imposed, a program could alter a procedure segment while executing in a high ring and plant in it code to execute later in a low ring (thus compromising the low ring). The third ring bracket is the _gate extension_ and is used to allow controlled entry to a lower-numbered ring. A process in a ring whose number is less than or equal to a segment's gate extension may call on the segment, but only at predefined gate entry points. Gate entry points (or just gates) may occupy the first locations in a procedure segment; the number of gates is specified by the _call limiter_ field in the segment's descriptor. When a procedure segment is entered at a gate, the processor's ring number changes to the number in the segment's read-execute bracket.

The gate mechanism is used by Multics user programs to invoke supervisor procedures. Each supervisor procedure has a series of gate entry points that correspond to that procedure's distinct functions. This mechanism, in contrast to the usual "supervisor call" allows the caller of a supervisor procedure to do (in a controlled way) the selection of a supervisor routine to perform a desired function. The gate extension allows a (user) program executing in ring four [3] to call on a supervisor program that must execute in ring zero. To take a specific example, the descriptor for a pure procedure supervisor segment readable by other supervisor

----

3 Multics historically allots rings zero through three to supervisor and supervisor support routines; thus user programs, by convention, execute in rings four or higher.

segments with five gates accessible by user programs (executing in rings 4 or higher) would contain the following information:

Read access bit: on
Write access bit: off
Execute access bit: on
Read-execute bracket: 0
Write bracket: 0
Gate extension: 4 or more
Call limiter: 5

User programs could only enter such a procedure at its five gates, each of which would (presumably) check to determine the legality of the service requested before performing it. Supervisor routines, on the other hand, could directly call internal entry points to the procedure if necessary since the processor would be in ring zero before entering the procedure in question.

The ring brackets and call limiter prevent less privileged routines from directly invoking more privileged ones in an uncontrolled way. However there is still the possibility of a less privileged routine passing as a parameter an address in a lower numbered ring and tricking a more privileged routine into using that address to write into a location that it (the privileged routine) can access but the calling routine cannot. This possibility is precluded in the Multics processor by associating ring numbers with each address formation register and with the stacks used for temporary storage and argument lists. The exact ring mechanism used in address formation and argument passing is detailed in [11]. The ring validation of arguments eliminates the "game of wits" that designers of most operating systems have been forced to play in order to eliminate the trickery mentioned above. No matter when an address supplied to the Multics supervisor is modified by a calling procedure, the calling procedure's ring number will be associated with the modified address, and prevent the supervisor from being tricked.

The Multics ring hardware would be of little use if the operating system allowed any user to construct a procedure segment with read-execute bracket of zero and gate extension of four--in effect, his own gate into ring zero. In fact, the supervisor routine that establishes ring brackets for a segment in the file system interprets a "validation level" computed from the current ring of execution of the process that creates the segment and allows the setting of ring brackets no lower than that level--four for an ordinary user's process. Special procedure segments provide gates for the setting of lower ring brackets. The access control lists for these segments are administratively controlled to allow execute access only to privileged system administrators and developers. Thus, the system can protect itself against the installation of unwanted gates and privileges.

Rings and Directory Segments

The ring brackets associated with directory segments are analogous to those associated with data segments. The read bracket controls status access to a directory, and the write bracket modify and append access. As the supervisor only allows itself the privilege of reading and writing in directory segments, all access to such segments actually occurs while the processor is in ring zero. However the supervisor routines that operate on directories compare the ring numbers of the routines that invoke them and the ring brackets of directories to assure that protection is enforced. As the operations on directories are performed interpretively by these supervisor routines rather than by the hardware executing user instructions, interpretive ring protection is efficient in the case of directory segments.

A second characteristic of ring protection and directories concerns the position of lower ring segments (of any type) in the file system hierarchy. A lower-ring segment may appear in the hierarchy subordinate to a directory with higher ring brackets. In particular, a process has status, modify and append access in ring four to its process directory which contains branches for a variety of process-related segments. Among these segments are the process' known segment table and process data segment which the process can only affect through ring zero supervisor routines. To prevent the process from altering or deleting such lower-ring segments while it is executing in a higher ring, the file system imposes an additional restriction--that access to branches in a directory is constrained by the ring brackets for the branches as well as those for the directory. Thus a user can list his process directory (status access), create a new segment subordinate to it (append), or delete such a segment (modify). However if he tries to replace, rename or delete a lower-ring segment such as his known segment table, the file system refuses his request. If the supervisor created a lower-ring directory segment subordinate to the process directory, the user would be restricted by the read (status) ring bracket for that directory from listing its contents, as discussed in the paragraph above. In addition, the restrictions discussed in this paragraph would prevent the user from deleting the directory and its subordinate segments.

Summary of Rings

Rings in Multics provide a powerful fundamental mechanism for isolating the supervisor and its data bases from damage or deception by user programs. This feature is of particular importance in the treatment of argument addresses passed to the supervisor.

In essentially every other system the supervisor is protected only by some primitive form of privileged mode (called master mode, supervisor mode, privileged mode, etc.) Each supervisor routine must inspect arguments provided by a user program, not only validating the argument values but also considering rather subtle factors such as user arguments passed "second hand" by other supervisor routines and the potential modification of arguments after they are checked but before they are used. The net result is an ad hoc "game of wits" between the designer trying to think of all the possible bad arguments, and the penetrator searching for one case the designer overlooked.

In contrast, the ring mechanism (as an adjunct to segmentation) provides a fundamental argument validation mechanism that is implicit regardless of how or when an argument is used. With rings, an execution time check is completely based on the <u>form</u> (ring brackets) rather than the <u>value</u> (memory location) of arguments. Thus this one mechanism provides a basic solution to a number of argument related vulnerabilities that is simply not attainable with the other hardware architectures commonly used today.

In summary, the use of protection rings in Multics provides:

a. Execution time validation of arguments from less privileged programs to more privileged programs.

b. Program controlled (interpretive) access to privileged data, e.g., directory segments.

c. An environment for support of user created privileged subsystems, e.g., a data base management subsystem.

Memory Management--Demand Paging

The discussion of virtual memory above pointed out that a segment is the smallest distinct entity of information subject to access control and distinguishable by the two dimensional addressing (segment number, segment offset) hardware. However, a segment is too large (64K words on the 645 and 256K words on the Level 68) to use as a single entity in the management of primary (core) and secondary (bulk store, disk, drum, etc.) memory. This problem is solved in Multics by the use of "demand paging", a scheme of growing popularity in recent years. Paging is basically independent of the access control mechanisms and, therefore, will not be discussed in detail. However, paging in Multics does provide a very efficient solution to the security problems involving information residues.

Paging Mechanism

Each segment in Multics is divided into a number of small blocks or pages of uniform size (currently 1024 words each). The directory branch for each segment includes a table that contains the location (storage device and address within the device) of each page of the segment. For segments currently being used (called "active segments" in the Multics jargon) a page table is created in primary (core) memory and accessed directly on each reference by the processor hardware to a segment. If the page is in primary memory the hardware completes the reference; if not, it traps to a supervisor routine that moves the page from secondary to primary memory, makes it accessible to all processes with that segment known, and then allows the processor to complete the reference. This memory management is invisible to any user's program. The details of the Multics paging mechanism are documented elsewhere [8].

Demand paging in Multics (and other systems) is an effective memory management scheme. All user programs and most supervisor programs are totally independent of the number, size, and type of primary and secondary storage devices. User programs have no need for "overlay" schemes to manage the memory accessible to them. Only those specific pages actually used (referenced) are brought into primary memory, and pages not actually modified in primary memory are not moved back to secondary storage (since a valid copy is already there). The end result is that each user has for his use a very large (virtual) core that usually far exceeds the real (hardware) primary memory available. This large available memory is particularly useful in interactive and terminal based systems in which the primary memory demands from user actions are difficult to predict and provide for in advance.

Although not directly part of the access control mechanism, demand paging has two significant security implications. First, the page transfer programs deal directly with real, rather than virtual resources--in particular absolute addresses. Therefore, these routines must be correctly implemented to prevent information spillage that would result from including an incorrect page in a user's segment. Experientially, such "misroutes" are not in evidence in Multics.

The second security implication of paging is totally pragmatic. Many contemporary operating systems have deep-rooted security flaws resulting from designs intended to overcome the limitations of the small, fixed blocks of memory available for use by the operating system. Typically, when more space is needed, parameters, buffers, or system routines will temporarily "borrow" user memory space and thereby introduce various security vulnerabilities. In Multics, such design approaches are avoided because individual segments can grow independently (viz., one segment never grows into another) and demand paging provides the operating system with the primary memory required.

Residue Control

The control of residues (information left behind in storage by some other user) is a major security problem in many systems. Multics has a basic and efficient solution to this problem. The usual ad hoc solutions to this problem involve overwriting (primary and secondary) memory when it is taken from one user and/or when it is given to a new user. Experience has shown that this approach has two major problems:

(a) It is practically impossible to insure that all the instances of residues are found and eliminated for all cases; and

(b) It can be highly detrimental to efficiency to be continually clearing all primary and secondary memory that is allocated to all users.

Because Multics has both segmentation and paging, the residue problem is solved in an effective manner. The previous discussion of segmentation showed that (because of the segmentation hardware) the only memory a user can access is that in a segment which he is authorized to use. When a Multics segment is created it is completely filled with zeros and throughout its life it will only contain information that authorized users have placed in it. Thus segmentation provides a fundamental solution to the problem of residue.

It would be a very inefficient use of secondary storage if the large number of segments containing mostly zeros were stored directly. Fortunately paging provides a mechanism to solve this practical problem. Each segment has a page map giving the actual location of each page. For any page containing all zeros, Multics uses a "null address" and the zeros are never actually stored on secondary storage. When such a page is actually referenced, it obviously cannot be transferred into primary memory from secondary storage, so the assigned page in core is actually cleared (set to zeros). This infrequent occurrence is the only condition under which a residue is actually cleared. All other core memory residues are automatically overwritten (when the core block is reassigned) by pages read in from secondary storage. Similarily, all secondary storage residues are automatically overwritten (when the storage is assigned to a previously all-zero page) by pages written out from core.

Summary of Memory Management

Multics uses demand paging to manage its real (physical) memory resources. Although paging is not directly a part of the access control mechanism its correct implementation is necessary to avoid misrouting of information. In addition, paging supports security by providing a mechanism that avoids ad hoc (and vulnerability prone) operating system storage management. Paging also provides an effective solution to the problem of memory residues.

THE MULTICS INPUT/OUTPUT SYSTEM

The design of the Multics input/output system supports the same access control concepts that have guided the other protection-related elements of the system. The input/output (I/O) area of computer systems has traditionally been a highly vulnerable one. Multics has avoided past design flaws by subjecting I/O operations to the access constraints of the virtual memory, by drawing a fundamental distinction between internal and external I/O [5], and by assuring that external I/O is exclusively symbolic (to "virtual" devices).

Internal I/O

The fundamental concept of information storage in Multics is that all of the storage media maintained by the computer facility should be managed by Multics rather than the individual users. Users then access all information as segments of the virtual memory and are not concerned with specific storage locations on (e.g.) tapes or disk packs. The internal I/O used to create the virtual memory is invisible to the user, and is part of the segmentation and paging mechanisms discussed above.

The Multics internal information storage is organized into a hierarchy according to storage device access speed and storage cost. The Multics design allows a user to influence the speed and cost of the storage media that hold his segments, but not to specify physical storage locations or to write his own programs to perform the input/output. The user and user programs have no direct access to specific media volumes or addresses, so the operating system need never perform ad hoc interpretive checking of user-specified I/O requests. Internal I/O is always requested implicitly when the user's program addresses a location in a segment. Access control over internal I/O is provided by the virtual memory mechanisms described above. The programs within the Multics supervisor that perform internal I/O are simple--they need only read or write fixed-length pages to or from secondary storage.

For large amounts of information (as anticipated by the Air Force Data Services Center) there is a practical (economic) need to store much of the information on demountable media such as disk packs. Access times to these portions of the virtual memory would, of course, be potentially long since the access times might include the time required for an operator to obtain and mount the media (just as in the more conventional case in which a user directly stores his information through explicit I/O to a tape or disk pack). A major functional limitation of the present Multics is that demountable internal I/O media are not yet implemented.

External I/O

One of the most powerful characteristics of the Multics design is the totally symbolic nature of the external I/O. External I/O is that input/output which a user requests to transfer information between his directly addressable (internal) virtual memory and an external medium that can, for example, be read, punched, or carried away. The cornerstone of protection in the Multics external I/O system is that the user is given no interface for directly controlling access to real devices. That is, the user cannot provide the conventional (and usually vulnerable) "device control words" or "channel programs". Although Multics does support the classical user I/O requests (like "read" and "write" in the FORTRAN language) these requests are always met through symbolically named "I/O streams" to what are effectively virtual devices. The user's only association with real devices is through the "attachment" of a logical device (e.g., a tape drive) which is part of the initial setup of each I/O stream for a process.

An implication of this virtual device approach is that external I/O is necessarily interpretive in that the operating system is invoked to translate each I/O operation requested by the user into real hardware commands. While this interpretation is essential to security (at least with current I/O architectures) it could be argued that the interpretation of external I/O imposes an unacceptable inefficiency on system operation. Fortunately, the efficiency problem is minimal because of the limited use in Multics for external I/O. The fundamental concept in Multics is that information being processed will be stored in the virtual memory and accessed through the efficient internal I/O mechanisms. Thus, in Multics explicit user (external) I/O is used only for the movement of information into or out of the facility--for example to terminals or to "stranger tapes" used to exchange information with other installations.

The implementation of the external I/O interface to real devices, unlike the design, is the one area of Multics that seems to have no simple underlying concepts for protection. Calls to the operating system for I/O services are implemented in much the same manner as similar functions in other contemporary systems. A penetrator can expect to find areas in which the designers failed to make an adequate check, and exploit the resulting vulnerability. Because much of the Multics external I/O system is rarely used, it tends to include much old and poorly understood code. Until there is a sound reimplementation of this area, it does not seem prudent, even in a controlled multilevel environment, to permit general user access to the gates that implement the interface to external I/O devices.

Summary of I/O

The Multics design provides an approach to I/O that can give effective protection, even with the lack of hardware support for protection in the contemporary I/O controllers and devices. The design for external I/O controls access by being totally interpretive and symbolic. The impact of this approach on efficiency is mitigated by reducing the role of external I/O in Multics, and by including all information in the system as part of the directly addressable virtual memory (even demountable media can be included in the virtual memory, and will be included in future versions of Multics).

CONCLUSION

The Multics design provides for extremely widespread and effective protection by applying a simple concept: The user operates in a completely virtual environment. Information accessible to the user, either in main memory or the file system, is included in a unified segmented virtual memory. Every segment has associated with it well-defined access rights for every user, and the hardware and operating system cooperate to enforce the access rights and restrictions. The virtual memory includes internal (file) I/O and completely hides internal I/O operations from the user. The paging mechanism used to implement the virtual memory eliminates "residue" problems in a complete and efficient way. External I/O is performed in terms of virtual devices and addresses in segments. The operating system has a basis for completely checking the legality of external I/O requests. The protection ring mechanism provides a flexible basis for isolating user and supervisor programs and data. In particular, it allows the operating system to detect, on a systematic basis, attempts to "fool" the operating system with bad addresses.

The major residual weaknesses of Multics lie in its implementation. The Honeywell 645 Multics processor had no hardware support for protection rings--instead, software was used to provide protections rings interpretively. The attempt to provide protection without excessive interpreting overhead resulted--as will be seen in the next section--in some vulnerabilities. While the Level 68 processor does have ring hardware, the transition from old to new processors also offers the possibility of implementation errors. The lack of a demountable segment mechanism in the current Multics results in an extra cost for on-line disk packs and drives in a facility that must handle many large files. Finally, the external I/O system, while conceptually simple, requires a complex implementation with today's hardware. The resulting complexity offers adequate chances for security-related errors and vulnerabilities.

A final point about the Multics design concerns its use in DoD classified processing environments. The basic Multics operating system includes no concept of classification or clearance. Such a concept must be devised and implemented if Multics is to support users with differing clearances in processing various levels of classified information.

In sum, the Multics design offers a sound basis for effective security. The major design deficiency is the lack of a notion of DoD classification and clearance, and it appears that this lack can be remedied in a straightforward way. Some implementation problems remain and, in one case (viz. external I/O), a solution will have to await the introduction of new improved hardware.

SECTION 4

SECURITY AND VULNERABILITY OF 645 MULTICS

INTRODUCTION

This section discusses the results of the vulnerability analysis of Multics for the Honeywell 645. The approach used in the vulnerability analysis was to identify general areas of the system and hypothesize a class of vulnerability in each area. Then that class was explored until one exploitable vulnerability was identified. The vulnerability and its exploitation were then demonstrated. No attempt was made to pursue the exhaustive search for "every" vulnerability that would be required by an attempt to "secure" Multics for use in an open multilevel environment.

The subsections below address the vulnerabilities that were identified in the three major areas considered; hardware, software, and procedures. More detail on each area is contained in Volume II by Karger and Schell [13]. A final subsection below summarizes the resources that were expended during the vulnerability analysis and the significance of the results obtained.

HARDWARE VULNERABILITIES

A traditional concern among developers of secure computer systems has been the possibility that a hardware failure would render the computer's access controls invalid, opening the way for a would-be penetrator to access any information in the system [14]. An approach to eliminating such vulnerability has been to develop a "subverter" program that runs periodically to ascertain that the nominal hardware controls are still present. A subverter program was developed and run on the Honeywell 645 at MIT in an attempt to detect such failures if any occurred. The following paragraphs discuss the results obtained by the subverter program.

Probabilistic Failures

The subverter program for the Honeywell 645 was executed every minute during 1100 hours of operation spread at random over about one year. The subverter attempted to execute privileged and illegal (undefined) instructions, to access segments in unauthorized ways, and to perform illegal operations using some of the complex execute instructions and address modifiers of the 645. During 1100 hours of operation, no security failures resulting from random hardware failures were detected.

Algorithmic Failures

While the subverter did not detect any random failures, it did produce two interesting results. Both were algorithmic--that is, they can be reproduced by any 645 program that chooses to invoke them at any time. The first subverter result was an undocumented instruction. One of the 645 instructions (octal 471) which, according to the processor manual is undefined, stores a word in the location addressed by the instruction. The makeup and origin of the word stored are not clear--the instruction has no discernable impact on system security.

The second algorithmic result was more interesting. In developing the subverter, special emphasis was placed on exercising the execute instructions and addressing modes as these seemed to require correct operation of very complex processor hardware. During the operation of the subverter it was found that an obscure combination of execute instruction and address modifiers, when placed in the correct locations in two segments, would completely bypass all access controls. A process using this combination could read or write any segment for which it had a descriptor. This power is sufficient, when exploited, to allow undetected access to any information stored by the computer.

The existence of this "indirect instruction" vulnerability seems to stem from a field hardware change made by Honeywell to all 645 processors. It appears that a rewiring of the processor for some entirely valid purpose had the side effect of eliminating the access controls in one obscure case.

SOFTWARE VULNERABILITIES

A traditional example of a software vulnerability in computer security is a vulnerability that allows a user to write a program that gains control of the processor in supervisor (master) mode. The 645 processor for Multics has an ordinary master mode like that of the 635 processor from which it is derived--in the 645 a master mode program can access in any mode any segment for which its process has a descriptor, no matter what access rights are specified by the descriptor. The protection rings of Multics are implemented on the 645 by software that supplies a separate descriptor segment with the appropriate access rights for each ring and that detects a "directed fault" and switches descriptor segments whenever a process switches rings. As will be seen in the next subsection, a complete penetration of Multics (access to any information) could be achieved by any program that obtained control of the processor in master mode or obtained arbitrary access to ring zero information. The following paragraphs discuss the three software vulnerabilities that were detected and brought to the point of exploitability. Methods of exploitation are discussed in the next subsection.

Argument Validation

Because the protection ring mechanism in 645 Multics is implemented by software, programs are required to perform several checks that are implemented by hardware in the follow-on Honeywell Level 68. One such check concerns the validity of arguments passed to the supervisor (or more generally from an outer ring to an inner ring). As the inner ring procedure has access to segments of its own ring, it must assure that it only reads or writes those segments in accord with its own purpose--it must not be "fooled" by a request from the outer ring.

A classic way to fool a supervisor into modifying itself or its data base is to pass it an argument pointer that addresses (possibly through a chain of indirect address pointers) supervisor storage. In Multics for the 645 a procedure, the argument validator, completed the indirect reference for each argument and checked the descriptor segment to assure that the calling procedure had the desired access to the argument finally addressed. Unfortunately the interpretation provided by the argument validator failed to take account of the fact that certain 645 indirect address words are modified by the hardware whenever they are used. Thus, it was possible to construct a chain of indirect words that was valid when accessed by the argument validator, but not when used by the called ring zero procedure. Supervisor procedures that read or write one of their arguments into another (for example) could thus be called and used to read or write any segment accessible by a process in ring zero.

The argument validation vulnerability was recognized at MIT and corrected by Release 18 of Multics. However, the existence and exploitability of the vulnerability were demonstrated on the Multics at Rome Air Development Center before Release 18 was installed there.

Master Mode Transfer

As was mentioned above, master mode is a hardware artifact of the 645 processor while ring zero is a creature of the Multics software for the 645. Control of the procedures designated master mode (by a bit in their segment descriptor words) is important since those procedures can access in any mode any segment for which there is a descriptor. A design convention for Multics initially required that every master mode procedure execute in ring zero. However, the cost of the software that interpreted the "crossing" or transition from ring to ring was relatively high and one master mode procedure--the fault signaller--was allowed to execute in any ring in an attempt to provide improved performance.

Programs executing in the same ring as the signaller were still restrained from gaining arbitrary access to it, for a slave mode procedure may only transfer to word zero of one that executes in master mode. Code at the beginning of the master mode procedure is expected to prevent the procedure from being "fooled" into an improper action. The signaller procedure evaluates the function it is requested to perform and, if that function is improper, it assumes that a system failure has occurred and transfers to a routine that shuts Multics down. Thus, a user can effect a system crash by calling the signaller with a bad argument.

A worse vulnerability is also associated with the signaller. The transfer to a shutdown routine referred to above is directed to an address defined by one of the processor address registers (the "linkage pointer"). By loading the linkage pointer with the address of an arbitrary instruction in a master mode segment, then calling the signaller with a bad argument, a program can cause the execution of the arbitrary instruction in master mode.

Exploiting the vulnerability outlined involved finding a master mode instruction that had some "useful" effect and that was followed by a transfer (or some such) instruction that would bring control back to the exploiting program. An instruction that did a load or store from (or to) an address specified by a pointer register was ideal. A load instruction with the desired environment was found in the signaller and a program written to test the exploitation. Unfortunately, an old listing of the signaller was used to locate the instruction. When the exploitation was tried, the transfer was directed to an LDBR (Load Descriptor Base Register) instruction that in effect isolated Multics from its entire virtual memory. The system crashed but the memory dump taken to diagnose the crash proved unreadable because of the absence of any virtual memory. The crash was attributed by the MIT system programmers to a disk hardware failure, and the ESD/MITRE analysis team decided that the master mode transfer approach, while exploitable, should be abandoned as too risky without complete documentation.

Unlocked Stack Base Register

Another vulnerability was associated with the signaller procedure and this, unlike the master mode transfer, was brought to the point of exploitation. The pointer registers in the processor can be "locked" by a master mode procedure so that a slave mode procedure cannot alter them. At one time, in the design of the Multics operating system, the "stack pointer" register was locked so that it would always point to the stack segment used for argument lists, temporary storage and so on. As the system design progressed it became clear that the inability to change stack segments was an unacceptable limitation for some application subsystems, and the stack pointer register was unlocked. At that time an "audit" of the supervisor was made to detect and correct programs whose integrity depended on the locked stack pointer.

The ESD/MITRE analysis team assumed that the original audit was incomplete and conducted one of their own. It was found that the signaller routine, before shutting the system down, stored all the processor registers on top of the stack. Changing the stack pointer would change the location where the registers were stored and, since the signaller was a master mode procedure, that location could be anywhere, even in another master mode segment.

Exploitation of the unlocked stack pointer involved a two-step procedure. First the processor registers were loaded with a sequence of instructions and the stack pointer was changed to point to a little-used master-mode procedure. The linkage pointer was set to cause a return to the exploiting procedure and the signaller was called with a bad argument. On return from the signaller the instruction sequence--an "execute double" instruction and a transfer to the exploiting procedure--had been stored in the master mode procedure. Then a processor pointer register was set to designate the two instructions to be executed in master mode and the linkage pointer was set to point to the newly installed instructions in the master mode procedure. A second call to the signaller with a bad argument, and any two instructions could be executed with master mode access privileges. These two instructions could read or write any segment for which the process had a descriptor accessible in the ring of execution of the signaller and the calling procedure. (Ring four in the case of a penetration.)

This vulnerability was discovered and code to exploit it developed during the fall of 1972. Because the exploitation involved executing experimental programs in master mode, and because it was desired to avoid crashing the MIT Multics, the programs were developed and tested at Rome Air Development Center. The programs performed satisfactorily and no crashes were caused at Rome.

However typing errors were made in the course of moving the programs from Rome to MIT and two crashes of the MIT system resulted. Although the MIT system programmers were aware that something was amiss, they did not detect the presence of a penetration or its objectives, and the exploitation proceeded successfully at MIT.

PROCEDURAL VULNERABILITIES

The hardware and software vulnerabilities outlined above provide a basis for obtaining complete control of a 645 Multics system and complete access to any information it stores. The details of gaining such control and access depend on the exact structure of the Multics operating system and on the specific control or access desired. The following paragraphs give a very brief overview of the methods that were used to exploit the vulnerabilities.

Patch and Dump Utilities

The hardware and software vulnerabilities described above give a programmer the ability to read or write one word of a segment. As a first step in exploiting the vulnerabilities, utility programs were developed to write or read any word of any segment for which a descriptor was available. The utility programs provided no additional access to information but did incorporate sufficient checking to guard against inadvertent errors and system crashes.

Changing the Process Identifier

When a Multics process attempts to make a segment known, the operating system determines the validity of the attempt by comparing the access control list entries for the segment with the process identifier stored in the process data segment (PDS) for the process. The PDS for a process is a ring zero data base, but is accessible to master mode programs that operate in the user ring--such as the signaller. The process identifier in the PDS is the "unforgettable identifier" [15] for Multics and determines whether the Multics access controls will properly deny access to an unauthorized user. The ability to read or write any segment for which a descriptor was available provided by the vulnerabilities outlined above is thus a sufficient mechanism to forge the unforgeable identifier and allow a process to obtain valid access to any segment in the entire Multics hierarchy.

The patch and dump utility programs mentioned above were used to implement a change-process command that altered the process identifier in the PDS to that of any selected user-project combination. Before issuing the change-process command, the would-be penetrator had to provide his "new identity" with access to any routines needed to complete the exploitation--but this was easily and legally accomplished via the standard Multics "set-access control list" commands. After using the change-process command, the penetrator was free to access any segment that was of interest to him and was accessible to the user whose identity he had assumed. The typical mechanism used to retrieve information was to copy the desired information into a segment legally accessible to the penetrator, then erase the audit trails (discussed below), and return to the penetrator's legitimate identity. The likelihood of detection while using the changed identity was remote, for the internal consistency of the Multics operating system was preserved when the new identity was in use.

Erasing the Audit Trails

The most heavily audited area of Multics is that of login and process creation. The answering service process that handles user logins records instances of invalid identities and incorrect passwords, and the identities of terminals from which they originate. Certain events are logged as they occur on the Multics operator's console as well as on-line storage. The system even provides each user who logs in with a note of the time of his last login and the terminal where it originated. Thus, a user can detect the event of a second individual logging in by stealing his password.

The audits surrounding the login process are both numerous and hard to evade. As a user has no process in existence during the time when the audit trails are being created, he cannot easily reverse them. Fortunately for the penetrator, the forged process identifier technique outlined above entirely bypasses the login auditing. The penetrator logs in as himself, then makes an internal change to a field of the PDS that is not used in auditing.

The approach outlined does, however, trigger one of the auditing operations built into Multics. Whenever a process accesses a segment it causes the "date-time used", "date-time modified", and "date-time entry modified" values in the directory entry for the segment to be updated as appropriate. A user listing a directory containing a sensitive segment might notice that the segment had been used or modified at an unusual time, and thus become suspicious. Fortunately, Multics provides tools to eliminate this audit trail. Commands are provided to reset each of the fields identified above. (The reason for providing these fields has to do with the need to restore the consistency of the file system after it has been "backed up" from tape). These commands can be used by any process with modify access to the directory containing the sensitive segment. Thus, it was only necessary for the penetrator to record each of the three sensitive date-time combinations before copying (or modifying) the target segment, and to restore each with the system-provided commands before resuming his own process identifier.

The Password File

In past vulnerability analyses of computer systems, the file of login passwords has typically been the prime target--it was clear that a penetrator in possession of this file could access any information in the system. In the case of Multics, the password file is not especially valuable, because the approach already discussed allows access to any information in the system without raising the auditing problems that appear when one logs in with another user's password. However, to demonstrate the completeness of the penetration, it was decided to obtain a copy of the Multics password file.

Copying the Multics password file was easy, given the tools described. Passwords are stored in a segment called the "person-name-table" (PNT) that is accessible to users on the SysAdmin (System Administrator) project. Thus it was merely necessary to change process identity to that of a SysAdmin user, copy the PNT, restore its date-time used, and restore the penetrator's identity.

A minor complication associated with copying the PNT was the fact that passwords in the PNT were enciphered. Although the cipher was supposed to be one-way (that is, transformed passwords in the PNT were compared to the result of the same transform applied to the user's login password) an inverse was discovered and, with some programming effort, implemented [16]. As a result, for a period of some six months, the ESD/MITRE vulnerability analysis team had the only cleartext copy of the MIT Multics password file.

(It should be noted that, even had the cipher in fact been one-way, the passwords could still have been recovered. In this case, the penetrator could have "bugged" the routine that encoded the passwords to make a clear text copy of each password in a file of his (the penetrator's) choosing. This latter approach is the general counter to all encryption schemes applied in systems with incomplete access controls.)

Planting Trapdoors

Each of the vulnerabilities outlined above has a significant disadvantage for the penetrator: it can be "fixed". The hardware access controls could be rewired, the signaller moved to ring zero, and so on. A penetrator who wished to obtain long-term access to a Multics facility would be well advised to take additional steps to assure his continued access. These steps can easily include the planting of trapdoors--small blocks of code in strategic programs that, when called with "key" arguments, do the penetrator's bidding in ring zero or in master mode.

Trap doors were planted in the MIT Multics programs by the ESD/MITRE analysis team. Trapdoors in object programs were used to insure guaranteed access if the original holes were fixed, and to increase the convenience of changing the process identity. Such object code trapdoors are hard to detect, but they "vanish" when the system is recompiled. Source language trap doors, on the other hand, "appear" when the system is recompiled and, if the facility under attack is the system development facility (as MIT is for Multics), they may even be distributed to other sites. Foreign code was even installed in a 6180 Multics program to demonstrate the ability to propagate vulnerability forward to a new system _in which all of the original vulnerabilities were corrected_.

RESOURCES AND RESULTS

This subsection summarizes the resources that were required during the vulnerability analysis of 645 Multics. It then goes on to discuss the significance of the results that were achieved by the analysis team.

Resources

Discussions of computer security frequently bring forth the point that "there is no absolute security" and that "one must consider costs and benefits to the penetrator". Such discussions are most convincing when pursued without hard data. To achieve the results outlined in this section (except that of deciphering the password file) required about two man-months of effort and less than $2000 in computer time. These results provide a penetrator with complete access to any information in any 645 (or, via a trapdoor, 6180) Multics, and with the ability to repeat such access arbitrarily often without danger of detection. A trapdoor installed in 6180 Multics would allow the installation at MIT and distribution of further trapdoors to provide more vulnerabilities in 6180 Multics.

Significance of Results

A direct conclusion supported by the vulnerability analysis reported here is that 645 Multics is not secure. Further, it is clear that 6180 and Level 68 Multics is jeopardized by the problems with 645 Multics, for if the ESD/MITRE team could plant trapdoors for distribution, others might also.

A more interesting conclusion is revealed by reviewing the nature of the penetration mechanisms discovered. Each was based on a compromise of an originally sound design. The signaller was moved to the user ring and a vulnerability introduced; the stack pointer was unlocked and a vulnerability introduced; a short cut was taken in argument list checking and a vulnerability introduced; a hardware flaw was "fixed" and a vulnerability introduced. In each case, the original decisions were made as part of a coherent overall design, then modified to achieve a localized improvements.[4] Further, the ESD/MITRE analysis team was drawn in its review of 645 Multics to exactly those areas where compromises were known to have been made. The explicit structure of the Multics operating system resulting from the use of segmentation and an underlying security design was a major aid in the search for vulnerabilities. (A corollary is that there is little excuse for these errors not having previously been found and corrected.)

The paragraph above is not intended to say that, had the 645 Multics implementation stayed with its original design, a secure system would have resulted. There was not, even in that design, a complete set of security principles and a proof of their sufficiency. However, security was a fundamental consideration in the Multics design and that design should have had few vulnerabilities if implemented properly. By compromising the initial principles, the implementers placed reliance on ad hoc checks--on finding _every_ vulnerability while confronted with a penetrator who must find _any_ vulnerability--and fared no better than others who have pursued the same path.

A final significant result of the vulnerability analysis concerns trapdoors. The analysis demonstrated the feasibility of installing a trapdoor that would be distributed by Honeywell to every 6180 or Level 68 installation. This approach seems to be the most economical and least risky one available to a serious penetrator. While the analysis team demonstrated a trapdoor installation using software penetration, a hostile agent might take advantage of the open development environment that supports Multics to achieve the same end.

----

[4] For completeness, it is noted that each of these vulnerabilities was corrected by the improved security hardware of the 6180.

SECTION 5

PROSPECTS FOR SECURITY OF 6180 MULTICS

INTRODUCTION

This section discusses the prospects for security in the Multics implementation for the Honeywell 6180 and Level 68.[5] The subsections below address in turn the issues of trap doors, vulnerabilities, and compliance with military security requirements. A final subsection presents conclusions on the overall security posture of 6180 and Level 68 Multics.

TRAP DOORS

The preceeding section described the relative ease of penetrating Multics on the 645 and of accessing any information stored in the system. Among the objects of information stored by the 645 Multics at MIT were the master copies of the source and object programs for Multics. It was on this installation that the initial Multics programs for the 6180 were developed and the initial system tapes for the 6180 produced. Thus, a penetrator who attacked the 645 could have installed trap doors in code that would later run as the 6180 Multics operating system--assuring himself access even if the design and implementation of 6180 Multics were perfect.

The hypothesis that trap doors are present in 6180 Multics seems a far fetched one--even to the author of this report. To appear in perspective, this hypothesis must be addressed in two parts: the likelihood that a trap door would be planted and the likelihood that it would be detected. The following paragraphs discuss each prospect in turn.

The prospect of a hostile or merely mischievous individual inserting a trap door into 6180 Multics is hard to assess. The problem is one of an intelligence threat assessment. Suffice it to

----

[5] The instruction set and protection features of the 6180 and Level 68 processor are the same; the Level 68 uses more modern and faster circuits than the 6180. A description of the 6180 and Level 68 architecture is contained in [17].

say, however, that Multics was developed under ARPA sponsorship, and was a subject of interest to the military for some time prior to the AFDSC acquisition. Thus, if there were an active program of planting trapdoors by a hostile agency, Multics might be a logical target. Its status as a target might be enhanced by its reputation for security and integrity, which might lead one to speculate that it would at some point be used in a multilevel security environment.

The prospect of a trapdoor avoiding detection is somewhat easier to address. Code for a trapdoor may be made arbitrarily obscure and, to avoid detection, should be concealed in a module unlikely to undergo close inspection or frequent revision. Code in about the amount required for a trapdoor was inserted in the code for 6180 Multics by the ESD/MITRE team and had escaped detection through the date of drafting of this report (some ten months later). Thus, it is probably reasonable to assume that an individual with the ability to penetrate 645 Multics could insert in 6180 Multics a trap door capable of evading detection.

The only viable response to this trapdoor threat is to control the environment in which the system is used. This measure has the effect of restricting exploitation of the trapdoor to one of a known set of individuals--and one who possesses some level of security clearance. The benefit of the controlled environment in reducing system vulnerability is thus a significant one.

VULNERABILITIES

A discussion of vulnerabilities in 6180 Multics must be somewhat speculative for, as of the drafting of this report, no vulnerability analysis had been pursued on the 6180. However two facts are reasonably clear even before such an analysis is pursued:

(1) The vulnerabilities that were exploited in the penetration of the 645 are corrected by the ring hardware for the 6180; and

(2) It is very likely that other vulnerabilities are present in 6180 Multics, in areas where compromises were made between "clean" design and efficient, convenient, or quick implementation.

It should further be noted that even in the absence of compromises with design principles, one could only demonstrate that 6180 Multics were secure if the design principles were proven consistent with a (presumably formal) secure system model using a well-defined process to establish the required correspondence.

An area of particular interest in pursuing a vulnerability analysis of 6180 Multics will be the input-output system. As Section 3 pointed out, the use of totally symbolic--that is, virtual--input/output (no direct user commands to the I/O controller) is a powerful technique for security; however, the present implementation of the Multics input/output system is relatively conventional, and a fair degree of complexity is needed to support the translation from virtual memory objects to the real memory locations required by the input-output multiplexor. A penetration attempt aimed at exploiting this complexity should have a fair chance of success. No such effort was directed at the 645 Multics input-output system because much easier and more obvious ways of demonstrating the system's vulnerability were discovered.

As a footnote to the discussion of 6180 vulnerabilities, it should be noted that, in the initial production implementation of 6180 Multics, the "template descriptor segment" used by the operating system to build descriptor segments for new processes included a Ring 4 gate to a highly privileged supervisor segment. Thus, any process was allowed to call on those highly privileged routines contained in that segment--and to use them to crash the system, write on the operator's console, or (in general) reconstruct the operating system. A two-minute inspection of the listing of the template descriptor segment used by the Multics supervisor to build descriptor segments for users at login was adequate to discover this vulnerability, and it was brought to the point of limited exploitation [18]. Once system maintenance personnel became aware of the problem (after the exploitation was demonstrated) they detected and corrected this vulnerability.

MILITARY SECURITY REQUIREMENTS

A matter somewhat separate from that of system integrity concerns the suitability of Multics for the processing of government classified information. The concern in this area is for the presence of system features for handling the military system's security attributes of level, category or compartment, and need-to-know.

At present, Multics controls access to information on a need-to-know basis. The access control list (ACL) is a mechanism that allows a user's process to identify those processes that can have access to a segment. The controls over access to directories provide a well-defined hierarchical organization for controlling need-to-know. There is, however, no concept in Multics of security level or category.

There are a number of practical reasons why a computer system that will be used to handle classified information should explicitly allow for classification and category. One such reason is that a system user should be provided automated assistance in determining the sensitivity of computer products--if a report has been prepared from segments classified secret and confidential, that report is most likely secret itself. The recipient of the report should be made aware that it may be secret so that he can safeguard it as secret or review it for a lower classification.

A more important issue in a multilevel computer system concerns the granting of access to information and the so-called "trojan horse" problem. A user who has modify access to a directory in Multics may grant any Multics user access to any segment subordinate to that directory. Thus, a user with appropriate access could allow a secret cleared user to read a top secret segment. Presumably, a conscientious user having such access would not do so, but two further issues arise:

(1) A user granting access should know that the potential clearance of any process which he allows to access information is consistent with the classification of the information; and

(2) A user accessing information does so through a process composed of programs in execution. Those programs may set the ACL for a segment on his behalf--with or without his explicit knowledge. A so-called "trojan horse" [19] program written by an untrusted individual and used on a sensitive segment might set the ACL on the segment so that another process could access it--without the knowledge of the responsible user.

A straightforward solution to the problems outlined above is to introduce the concepts of classification and compartment into Multics. Classification and compartment attributes can be applied to processes, segments, directories, terminals and interprocess communication channels. Users can be assigned clearances and a suitable set of algorithms introduced into the operating system to interpret the security attributes and allow or deny access as appropriate. A basis for these algorithms, with sufficient power to render "trojan horses" ineffective, has already been defined by Bell [20] and Walter [21].

CONCLUSION

The paragraphs above have discussed the probable security posture of Multics for the Honeywell 6180 and Level 68. The discussion, while fairly general, points to the conclusion that a hostile agent should be able to develop a program to penetrate the current 6180 Multics. Further, there appears to be a possibility that a "trap door" could be present in (or introduced into) 6180 Multics. In either case, with the widening availability of Multics computers, it should be possible for a penetrator to develop his tools well away from AFDSC and bring them to an AFDSC Multics in a form ready to run with little time and no noticeable impact.

On a less speculative front, it is clear that 6180 Multics in its present form does not include features to aid in the management of government classified information or to prevent the operation of "trojan horses". For Multics to be used to process multiple levels of classified information in a convenient and controlled way, such features should be added.

Section 7 presents a series of recommendations for action aimed at making an AFDSC Multics usable in a controlled multilevel security mode. These recommendations are based on the results and conclusions of this section and Section 4.

SECTION 6

LONG-TERM PROSPECTS FOR SECURITY IN MULTICS

INTRODUCTION

This section discusses the long-term prospects for implementing a secure multilevel computer utility based on Multics. It begins with a general discussion of the principles that are the basis for implementing a certifiably secure computer system. Then follows a discussion of the applicability of these principles to the Multics hardware and operating system.

CERTIFIABLY SECURE SYSTEMS

Introduction

This subsection discusses the technical principles on which the development of a certifiably secure computer system can be based. The initial subsection below provides a bit of background to the development of the principles. Then follows a discussion of the basic requirements that a secure system's hardware and software must meet. The final subsection discusses the requirements imposed by the need for "certifiable security". Much of the material that follows has been taken from [22] which presents a more detailed picture of the intermediate development steps that can lead to a certifiably secure computer system based on Multics.

The Computer Security Technology Panel

In an attempt to determine the reasons for the impossibility of securing the GCOS III operating system at AFDSC, and to identify ways of solving future multilevel security problems, ESD in early 1972 convened a computer security technology planning study panel. The panel operated under a contract from ESD to James P. Anderson and Company and was directed to prepare a development plan representing a coherent approach to attacking the problems of multilevel computer security. The panel's report [3] identified the problem of completeness--that as long as even one security-related defect remains in a computer or operating system, that system provides absolutely no protection against a hostile penetrator (who may access any information in the system at will). The panel thus recognized the futility of "patching holes" in an operating system like GCOS III as an ad hoc search for absolute perfection.

The technical approach recommended by the panel was "to start with a statement of an ideal system, a model, and to move the statement through various levels of design into the mechanisms that implement the model system" [3], p. iii. The following paragraphs discuss the characteristics of the "ideal system" as proposed by the panel and detailed by subsequent efforts.

The Reference Monitor

The basic component of the ideal system proposed by the security technology panel is the reference monitor--an abstraction that controls the access of subjects (active system elements) to objects (units of information) within the computer system. Figure 5 presents a schematic diagram of the relation among subjects, objects, reference monitor, and reference monitor authorization data base. The figure gives examples of typical subjects, objects, and data base items.

In operation, an implementation of the reference monitor allows or forbids access by subjects to objects, making its decisions on the basis of subject identity, object identity, and security parameters of the subject and object. The reference monitor implementation both mechanizes the access rules of the military security system and assures that they are enforced within the computer.

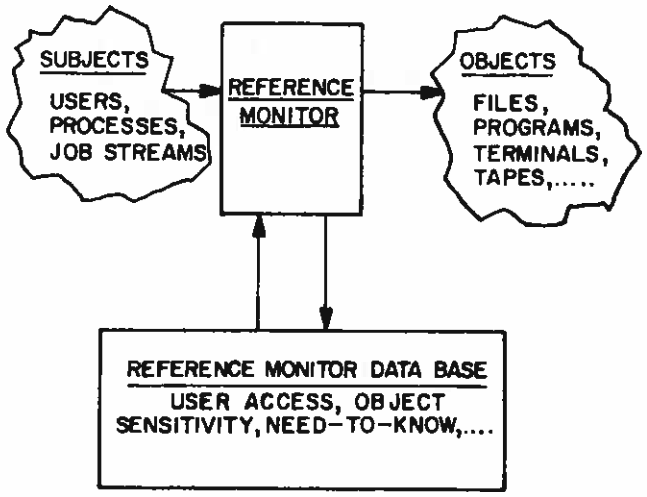
The security technology panel stated that a reference monitor implementation must meet the following three requirements in order to provide the basis for a multilevel secure computer system:

a. Completeness--the implementation must be invoked on every access by a subject to an object;

b. Isolation--the implementation and its data base must be protected from unauthorized alteration;

c. Certifiability--the implementation must be small, simple and understandable so that it can be tested and verified to perform its functions properly.

Both the requirement for completeness and that for certifiability demand that the mechanism that implements the reference monitor include hardware as well as software--the former because software validation of every access by a subject to an object would add intolerable complexity and overhead; the latter because certain


SUBJECTS
USERS, PROCESSES, JOB STREAMS
REFERENCE MONITOR
OBJECTS
FILES, PROGRAMS, TERMINALS, TAPES,.....
REFERENCE MONITOR DATA BASE
USER ACCESS, OBJECT SENSITIVITY, NEED-TO-KNOW,....

IA-42,282

Figure 5 REFERENCE MONITOR

hardware architectures preclude the construction of a simple understandable operating system. The hardware/software mechanism that implements the reference monitor has been called the "security kernel". When the hardware is specified, the software part of the implementation is sometimes referred to as the security kernel for that hardware.

Recognizing the importance to achieving computer security of the panel's ideal model of a reference monitor, ESD initiated the development of a mathematical model of computer security. Preliminary efforts were performed by ESD [22] and the initial model development was completed by The MITRE Corporation. A later modelling effort using an alternate approach has been pursued in parallel with the MITRE work by Case Western Reserve University.

The MITRE model [20] represents a secure computer system as a finite-state mechanism that makes explicit transitions from one security state to the next. The state of the system is defined by: (a) the classifications and formal categories of all subjects and objects; (b) the need-to-know relationships of subjects and objects; (c) the hierarchical organization of objects (in a storage system); and (d) the current ability of subjects to access objects. The model specifies rules that define formally the conditions under which a transition from state to state may occur. The rules are proven to allow only transitions that preserve the security of information in the system. A significant property of the model is the *-property that states that all but trusted programs are restricted from writing information at a lower classification (or proper subset of categories) than they read. The restriction prevents information obtained at the higher security level from being transferred to a lower level where it can be accessed illegally. This property eliminates the need to certify that programs such as editors and utility routines do not act as "trojan horses" and downgrade classified information.

The finite-state model specifies the secure operation of a system composed of subjects and objects. A security kernel must implement representations of both the rules of the model and the subjects and objects these rules control. The implementation of subjects and objects is constrained by the hardware on which the kernel operates. If the hardware does not facilitate the simple implementation of subjects and objects, the third of the panel's requirements for a reference monitor will not be met. The panel recognized this fact and recommended for secure computer systems the use of descriptor-driven processors that implement segmented memories. With such processors, the objects of the model can correspond to the segments supported by the hardware. A properly organized segmented memory that merges primary (core) and secondary storage management functions eliminates from security consideration any separate, complex, and security-related "file system". Further, the subjects of the model correspond to processes (address space-processor state pairs) supported directly by a descriptor-driven processor.

The security kernel defined by the model and implemented on descriptor-driven hardware is a simple software mechanism that implements only the security rules, subjects, and objects. It does not provide the full facilities of an operating system; it could not do so without developing so much complexity that it would no longer be a security kernel. Instead, the complex functions required of an operating system are provided by programs external to and controlled by th kernel. These functions can be arbitrarily complex but are not security related. However, some may be sensitive in terms of assuring the smooth operation of the computer system. For example, a typical operating system (not kernel) function like a scheduling algorithm cannot compromise information, but it can slow service to users.

To assure that user programs can be separated from (and kept from interfering with) such sensitive programs, the MITRE development efforts in multilevel security identified the need for hardware with at least three separate domains of execution (states of program privilege). Of these, one can be allocated to the kernel, the second to the operating system, and the third to user programs. The kernel can easily protect the operating system from user programs and, because of the organization of the hardware, the transitions from one domain to another can be rapid and efficient.

In summary, this subsection has identified the concepts of a security kernel and discussed a model of a kernel that represents the secure operation of an ideal reference monitor. It has also mentioned the requirement that the kernel implement subjects and objects, and pointed out that their simple implementation hinges on the architecture of the computer on which the kernel is based. In particular, secure computer hardware is required to provide a segmented memory and at least three processor domains. The above discussion has not been explicit about the transition from the model to programs that implement a kernel on specific hardware. A discussion of that transition is included in the next subsection.

Security Certification

A design for a security kernel to control suitable hardware is extremely small and simple, when compared to a design for a normal operating system. The design of a security kernel that can support general-purpose computing on a Digital Equipment Corporation

PDP-11/45 minicomputer, for example, can be implemented by a small (less than one thousand line) structured computer program, and follows the MITRE mathematical model directly [23]. However, it is still necessary to verify that the objects and subjects provided by the kernel are those specified by the model. An approach to providing this verification has been identified and is being pursued. This approach is based on the work of Price [24] as refined by Robinson, et al [25]. The approach involves preparing a formal specification for each function of the kernel and identifying those assumptions on which the correct operation of each function depends. The specifications of functions are proven consistent with the requirements of the security model. A proof is then constructed that demonstrates that all of the assumptions are preserved by all of the functions. Successive more detailed specifications for each function are constructed and proven to implement the first specification correctly. The final detailed specifications of functions are close to a programming language and facilitate proof or verification of the code that implements the specified kernel design.

In summary each step in the sequence from model to kernel code must be subject to proof or verification. The process of certification then becomes a positive one--the assessment of well-identified and reasoned proof steps--rather than an exhaustive and an exhausting quest for perfection in the operating system in which no constructive definition of perfection exists.

MULTICS AND CERTIFIABLE SECURITY

The following paragraphs discuss the prospects of Multics for supporting the development of a certifiably secure computer system based on the principles defined above. The initial discussion considers the 6180 and Level 68 Multics hardware; the Multics operating system and application programs are addressed by the second subsection below.

Hardware

The subsection above mentioned the need for a descriptor-driven segmented memory and for multiple domains of execution to support a certifiably secure system. The paged segmented virtual memory of the 6180 Multics processor meets the former requirement fully, and the protection rings provide an adequate mechanism for meeting the multiple domain requirement. Thus, the 6180 processor will support the implementation of a security kernel.

The input-output architecture of the 6180 is both more conventional and less hospitable to a kernel than that of the processor. While internal input-output is part of the operation of the segmented virtual memory and performed only by a security kernel, external input-output (viz. to a terminal) is performed at direct user request. It is in the specific area of external input-output that the 6180 hardware architecture raises concern. Fortunately, the 6180 uses both totally symbolic external I/O and a separate processor (a Datanet 355) to effect external input-output, and it appears feasible to replace this processor with a secure minicomputer with a kernel to provide a complete secure path for external input-output.

Software

Multics does not today include a security kernel. If a Multics operating system is to be certified as secure, such a kernel must be developed to control the 6180 hardware in compliance with a mathematical model of secure operation. Such a kernel must be developed "from scratch" though the hardware is already known to be suitable.

In the areas of non-kernel operating system software and of application software Multics has a significant advantage. The Multics operating system software is written to control and exist in a segmented virtual memory. Thus, it offers a substantial degree of compatibility with a security kernel environment. The operating system programs would have to be modified, not scrapped and rewritten, to coexist with a kernel. As the kernel is (by definition) a complete security control mechanism, the non-kernel operating system programs need not in any sense be "secured". As development planning estimates indicate that a kernel will not be as costly as the remainder of an operating system, the compatibility of Multics yields a considerable saving in the cost of developing a prototype of a secure system.

The saving for application programs should be even greater than that for the operating system. Nearly all application programs for Multics should run unaltered on a modified Multics with a kernel. This statement is made especially true if only symbolic I/O is permitted and if the notions of classification and category of the military security system are incorporated in the "near-term" 6180 Multics discussed in Section 5. Then only programs that attempt to perform inherently unsecure operations will be invalidated by a kernel.

CONCLUSION

Multics and the Honeywell 6180 and Level 68 provide a singularly appropriate foundation for a certifiably secure system. The technology to support the development of such a system is already known and relatively mature.

SECTION 7

RECOMMENDATIONS

INTRODUCTION

This section presents recommendations for the use of Multics at AFDSC. The recommendations are presented in three subsections dealing with near-term development efforts, near-term use of Multics, and longer-term development efforts.

NEAR-TERM DEVELOPMENTS

Sections 4 and 5 identified security weaknesses and deficiencies associated with the Multics operating systems for the 645 and 6180. If 6180 Multics is to be used in a controlled environment for the simultaneous processing of secret and top secret information, the following development steps are indicated:

(1) Conduct an analysis of 6180 Multics to identify existing vulnerabilities that can be found and direct that they be corrected. The explicit structure of the Multics security controls should make this analysis much more productive than those for more conventional systems.

(2) Eliminate from 6180 Multics those functions that comprise major areas of potential vulnerability and whose functionality can be provided in an alternate way. The major such area identified to date is direct user control of tape input-output. An alternative--system control of tape I/O by a process that provides a well-defined interface to user programs and is developed by cleared individuals--can provide the required functions.

(3) Add the attributes of the military security system (classification and compartment) to segments, directories, processes, and other objects managed by Multics. Use these attributes in controlling access, as specified by existing computer security models. Provide appropriate tools for controlling these attributes, including those required by a system security officer.

(4) Recompile the Multics operating system in the AFDSC environment to eliminate possible object code trap doors.

(5) Review for source program trap doors all directly security-related ring zero modules (such as those that create descriptors). Such a review will presumably be included in the vulnerability analysis of (1) above. In addition, review a random sample of (ostensibly) non-security related ring zero routines and routines used for system generation (such as those in the compiler) for source code trap doors.

(6) Provide an auditing capability to allow a system security officer to assure that the system is being used properly. Such auditing should cover actions by the security officer as well as users, and should allow the security officer some latitude in determining what user actions are audited, and when.

(7) A review of the 6180 operation codes, addressing, and timing should be conducted to detect possible errors in hardware design and implementation.

(8) A subverter program should be developed to detect possible hardware failures that might effect security. The subverter should be programmed to test different combinations of instructions, execute instructions, and addressing organization in an attempt to identify algorithmic hardware problems like the one in the 645.

(9) New features introduced to provide the AFDSC Multics with added utility such as embedded GCOS and batch processing features, should be developed within the basic Multics structure and should operate subject to all security and access controls.

As part of the planning for installation of a 6180 at AFDSC and its use in a controlled environment, Honeywell conducted, under Air Force contract, a design analysis that addresses the points raised under (2), (3), and (6) above [26].

SYSTEM USE

A Multics operating system that has undergone the modifications outlined above can be used in a controlled environment with secret and top secret users and information. There is a risk, in such a mode, that a secret-cleared user will "go bad" and, possibly in collusion with an outsider in the Multics development environment, begin to access top secret data. Recommendations aimed at preventing such an event are:

(1) The Multics software at AFDSC should be configuration-controlled and protected from modification "as though top secret".

(2) All new ring zero operating system modules and all new modules in the system generation software should be reviewed at AFDSC for trap doors and should be compiled at AFDSC.

(3) The system security officer at AFDSC should be aggressive. He should review the audit trails, monitor users' usage patterns, and periodically (and at random) monitor users' console sessions for suspicious activity.

(4) Under no circumstances should uncleared individuals be allowed to submit jobs to an AFDSC Multics, or to receive unscreened outputs. This restriction supports the controls on the system environment and limits the opportunity to exploit a trapdoor if one is present.

(5) The subverter program should be run whenever the AFDSC Multics is in operation. Any anomalies detected by the subverter should be analyzed until an exact cause can be determined.

(6) Hardware modifications to the 6180 (especially the CPU) should be reviewed and understood by cleared personnel before being installed, to avoid the introduction of hardware flaws or trap-doors.

LONG-TERM DEVELOPMENTS

In the longer term, the Multics hardware and operating system provide an attractive base for the development of a secure system capable of supporting open multilevel operation without the major risks of developing a completely new hardware-software system using a previously untried design. If such operation is required as a future capability by AFDSC, then AFDSC should:

(1) Express its requirements for such an open multilevel system through formal Air Force channels.

(2) Coordinate its use of Multics with long-term research and development efforts in order to provide a smooth transition from the near-term two-level Multics to the open multilevel system.

REFERENCES

1. C. W. Beardsley, "Is Your Computer Insecure?", IEEE Spectrum, January 1972, pp. 67-78.

2. L. Smith, "Architectures for Secure Computer Systems," ESD-TR-75-51, Bedford, Massachusetts,: The MITRE Corporation, April 1975.

3. J. P. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, Volume I, Fort Washington, Pennsylvania: James P. Anderson & Co., October 1972.

4. M. D. Shroeder, "Performance of the GE-645 Associative Memory While Multics is in Operation," Proc. ACM Workshop on System Performance Evaluation, Harvard University, April 1971, pp. 227-245.

5. E. L. Burke, "Concept of Operation for Handling I/O in a Secure Computer at the Air Force Data Services Center (AFDSC)," ESD-TR-74-113, Bedford, Massachusetts,: April 1974.

6. G. E. Reynolds, "Multics Security Evaluation: Exemplary Performance Under Demanding Workload," ESD-TR-74-193, Volume IV, Bedford, Massachusetts,: Air Force Electronics Systems Division, June 1974.

7. S. M. Goheen and C. D. Jordan, "Evaluation of TICS: A Multics Subsystem for Development and Use of CAI Courseware," ESD-TR-75-56, Bedford, Massachusetts,: The MITRE Corporation, 1975.

8. A. Bensoussan, C. T. Clingen and R. C. Daly, "The Multics Virtual Memory: Concepts and Design," Communications of the ACM, Volume 15, Number 5, May 1972, pp.308-318.

9. R. J. Feiertag and E. I. Organick, "The Multics Input/Output System," Proc. of ACM Third Symposium on Operating Systems Principles, Palo Alto, California, 1971, pp. 35-41.

REFERENCES (Continued)

10. E. I. Organick, The Multics System: An Examination of Its Structure, Cambridge, Massachusetts,: MIT Press, 1972.

11. M. D. Schroeder and J. D. Saltzer, "A Hardware Architecture for Implementing Protection Rings," Communications of the ACM, Volume 15, Number 3, 1972, pp. 157-170.

12. J. H. Saltzer, "Protection and the Control of Information in Multics," Communications of the ACM, Volume 17, Number 7, July 1974, pp. 388-420.

13. P. A. Karger and R. P. Schell, "Multics Security Evaluation: Vulnerability Analysis," ESD-TR-74-193, Volume 2, Bedford, Massachusetts,: Electronics Systems Division (AFSC), L. G. Hanscom Field, June 1974.

14. L. M. Molho, "Hardware Aspects of Secure Computing," Proceedings AFIPS 1970 SJCC, Montvale, New Jersey,: AFIPS Press, pp. 135-141.

15. B. W. Lampson, "Protection," Proc. Fifth Princeton Symposium on Information Sciences and Systems, Princeton University, March 1971, pp. 437-443. Reprinted in Operating Systems Review, 8,1, January 1974, pp. 18-24.

16. P. J. Downey, "Multics Security Evaluation: Password and File Encryption," ESD-TR-74-193, Volume III, Bedford, Massachusetts,: Electronic Systems Division, June 1974.

17. E. L. Burke, M. Gasser and W. L. Schiller, "Emulating a Honeywell 6180 Computer System," RADC-TR-74-137, Bedford, Massachusetts,: The MITRE Corporation, June 1974.

18. T. Alexander, "Waiting for the Great Computer Rip-Off," Fortune, Volume XC, Number 1, July 1974, pp. 142-150.

19. D. Bransted, "Privacy and Protection in Operating Systems," Computer, Volume 6, Number 1, January 1973, pp. 43-47.

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

21. K. G. Walter, W. F. Odgen, W. C. Rounds, F. T. Bradshaw, S. R. Ames, Jr., and D. G. Shumway, "Primitive Models for Computer Security," ESD-TR-74-117, Cleveland, Ohio,: Case Western Reserve University, January 1974.

22. R. R. Schell, P. J. Downey, and G. J. Popek, "Preliminary Notes on the Design of Secure Military Computer Systems," MCI-73-1, Bedford, Massachusetts,: Electronic Systems Division (AFSC), L. G. Hanscom Field, January 1973.

23. W. L. Schiller, "The Design and Specification of a Security Kernel for the PDP-11/45," ESD-TR-75-69, Bedford, Massachusetts,: The MITRE Corporation, May 1975.

24. W. R. Price, "Implications of a Virtual Memory Mechanism for Implementing Protection in a Family of Operating Systems," Ph.D. Thesis, Pittsburgh, Pennsylvania,: Carnegie-Mellon University, June 1973.

25. L. Robinson, P. G. Neumann, K. N. Levitt, and A. R. Saxena, "On Attaining Reliable Software for a Secure Operating System," 1975 International Conference on Reliable Software, Los Angeles, California, April 1975, pp. 267-284.

26. Honeywell Information Systems, "Design for Multics Security Enhancements," ESD-TR-74-176, Bedford, Massachusetts,: Electronic Systems Division (AFSC), L. G. Hanscom Field, December 1973.

DISTRIBUTION LIST

INTERNAL

D-70

J. J. Croke
W. S. Melahn
J. W. Shay

D-73

S. B. Lipner

D-75

S. R. Ames, Jr.
W. Amory
D. L. Baldauf
E. L. Burke
F. Chess
J. A. Clapp
T. L. Connors
M. J. Corasick
M. Ferdman
M. Gasser
J. B. Glore
M. Hazle
D. W. Lambert
L. J. LaPadula
J. K. Millen
D. G. Miller
G. H. Nibaldi
C. M. Sheehan
J. D. Tangney
P. S. Tasker
B. N. Wagner
P. T. Withington
J. P. L. Woodward

EXTERNAL

Electronic Systems Division
Hanscom Air Force Base
Bedford, MA 01731

TOI

Col. N. Michaud

TOIT

Lt. Col. C. J. Grewe
P. R. Veckery