FINAL EVALUATION REPORT
Digital Equipment Corporation

OpenVMS and SEVMS, Version 6.1, with VAX or Alpha

NATIONAL
COMPUTER SECURITY CENTER

9800 Savage Road
Fort George G. Meade
Maryland 20755-6000

24 October 1996

Library No. S-243,804
Final Evaluation Report Digital Open VMS and SEVMS, Version 6.1, with VAX or Alpha

The following are trademarks of Digital Equipment Corporation:

AlphaServer, BI, CI, DEC, DECchip, DECDtm, DECEvent, DEClaser, DECnet, DECwindows, DELNI, DELUA, DEQNA, DEUNA, DHB32, Digital, HSC, HSC40, HSC50, HSC60, HSC70, HSC90, HSD20, HSD30, HSJ, KDA, KDM, KLESI, KRQ50, LA, LAT, LN01, LN03, LN09, LP27, LQP02, LSI-11, LXY, MI, MicroVAX, MicroVAX II, MSCP, OpenVMS OpenVMS Cluster, PDP-11, Q-BUS, BA, RC, RS, RL, RP, RXD, RRD42, RRD90, RUX50, RV20, RX, RX01, RX02, RX211, RX23, RX33, RX50, RZ, SBI, SDI, STI, TA, TE, TK, TM, TM03, TM78, TMSCP, TQK, TQK50, TS, TS11, TU, UDA, UNIBUS, VAX, VAX 6000, VAX 8200, VAX 8250, VAX 8300, VAX 8350, VAX 8550, VAX 8600, VAX 9000, VAX-11/780, VAXB, VAXcluster, VAXserver, VAXstation, VAXstation 4000 VLC, VAX Volume Shadowing, VMS, VMScluster, VT, VT52, VT100, VT220, VT300, VT320, VT330, VT340, VT420.

FINAL: 24 October 1996
FOREWORD

This publication, the Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha is being issued by the National Computer Security Center under the authority of and in accordance with DoD Directive 5215.1, “Computer Security Evaluation Center.” The purpose of this report is to document the results of the formal evaluation of Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha operating system. The requirements stated in this report are taken from Department of Defense Trusted Computer System Evaluation Criteria, dated December 1985.

Approved:

John C. Davis 24 October 1996
Director,
National Computer Security Center
ACKNOWLEDGEMENTS

SA-Team Members

This RAMP Action involved a Security Analysis Team (SA-Team). SA-Team members included the following individuals:

Hal Bettle
Curtis J. Coppersmith
Digital Equipment Corporation
110 Spit Brook Road
Nashua, NH 03062

Frank Belvin
Timothy J. Bergendahl
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730

For their contributions to this RAMP Action, acknowledgement is given to James Reynolds of The MITRE Corporation and Mei-ying Lin, a Digital Equipment Corporation VSA.
# TABLE OF CONTENTS

## FOREWORD

iii

## ACKNOWLEDGEMENTS

v

## EXECUTIVE SUMMARY

xvii

## I System Overview

1

### 1 Introduction

1.1 Evaluation Process Overview .............................................. 3
1.2 Background and History .................................................. 5
1.3 Document Organization ................................................... 6
1.4 Conventions ................................................................. 6

### 2 TCB Overview

9

2.1 Software ................................................................. 10
2.2 Firmware ................................................................. 11
2.3 Hardware ................................................................. 11

### 3 Hardware Architecture

13

3.1 VAX Architecture .......................................................... 13
  3.1.1 Architecture Overview .............................................. 13
  3.1.2 Instruction Set ....................................................... 14
  3.1.3 Process Management ................................................ 22
  3.1.4 Interrupts and Exceptions ......................................... 23
  3.1.5 Memory Management ............................................... 26
  3.1.6 I/O Management ..................................................... 32
3.2 Alpha Architecture ........................................................ 34
  3.2.1 Architecture Overview .............................................. 34
  3.2.2 Registers and Instruction Set ...................................... 35
  3.2.3 Instruction Set and PALcode Environment ......................... 35
  3.2.4 Process Management ............................................... 40
  3.2.5 Interrupts, Exceptions, and Machine Checks ...................... 42
  3.2.6 Memory Management ............................................... 43
3.3 VAX or Alpha System Node ............................................... 47
  3.3.1 Buses ...................................................................... 47
  3.3.2 I/O Bus Adapters and Controllers .................................. 48
  3.3.3 Console Subsystems .................................................. 48
  3.3.4 Storage Devices ...................................................... 53
  3.3.5 Terminals and Printers .............................................. 54
  3.3.6 VAX Processors ...................................................... 54
  3.3.7 Alpha Processors ..................................................... 56
3.4 VMScIuster ......................................................... 57
     3.4.1 Configurations ............................................... 57
     3.4.2 Stand-alone System ........................................ 58
     3.4.3 The CI Cluster ............................................ 58
     3.4.4 The NI VMScIuster ........................................ 63
     3.4.5 The Mixed Interconnect VMScIuster ......................... 68

4 Software Architecture ........................................... 73
     4.1 Software Structure .......................................... 73
         4.1.1 Executive Images ....................................... 73
         4.1.2 Non-Executive Trusted Code ............................. 75
         4.1.3 Security Database ...................................... 78
         4.1.4 Untrusted Software Interface ........................... 79
         4.1.5 Privileged Interfaces ................................... 80
     4.2 Internal TCB Communication Mechanisms ....................... 80
         4.2.1 Interrupts and Spinlocks ................................ 81
         4.2.2 Mutual Exclusion Semaphore ............................ 82
         4.2.3 Interprocess Communication System Service .......... 82
         4.2.4 System Communication Architecture ..................... 83
         4.2.5 DEClnet ................................................ 85
     4.3 Memory Management ........................................... 87
         4.3.1 Pages .................................................. 87
         4.3.2 Sections ............................................... 89
         4.3.3 Images ............................................... 90
         4.3.4 Virtual Address Mapping ................................. 90
         4.3.5 Management Within a Virtual Address Space .......... 92
     4.4 Process Management ........................................... 98
         4.4.1 Process Data Structures ................................ 98
         4.4.2 Process Creation ........................................ 99
         4.4.3 Images ................................................ 99
         4.4.4 Process Deletion ....................................... 101
         4.4.5 Process Scheduling ..................................... 101
         4.4.6 Processes with a CLI ................................... 102
     4.5 Interprocess Communication .................................. 104
         4.5.1 Common Event Flag Clusters .............................. 104
         4.5.2 Logical Names ......................................... 105
         4.5.3 Mailboxes ............................................. 106
         4.5.4 Resource Locks ......................................... 106
         4.5.5 DEClnet ................................................ 109
         4.5.6 MAIL and PHONE Utilities ............................... 109
     4.6 File Management ............................................... 111
         4.6.1 Disk File Specification ................................ 112
         4.6.2 Disk Volumes ........................................... 112
         4.6.3 File Identification ..................................... 114
         4.6.4 Extended QIO Processor (XQP) .......................... 114
         4.6.5 Tape Volumes .......................................... 116
         4.6.6 Accessing Files In A Cluster ........................... 118
         4.6.7 Record Management Services ............................ 118
II Security Functionality

5 TCB Protected Resources

5.1 Subjects .................................................. 135
  5.1.1 Access Rights Block .................................. 135
  5.1.2 User Name ............................................. 135
  5.1.3 User Identification Code ............................. 137
  5.1.4 Identifiers and Rights Lists ......................... 137
  5.1.5 SEVMS Subject Classification ....................... 137
  5.1.6 Privileges ............................................. 138
  5.1.7 Propagation of Process Security Attributes ...... 142

5.2 Objects .................................................. 143
  5.2.1 Common Object Attributes .......................... 143
  5.2.2 Device Objects ........................................ 146
  5.2.3 File System Objects .................................. 148
  5.2.4 Memory-resident Objects ......................... 150
  5.2.5 Queue Objects ....................................... 153

6 TCB Protection Mechanisms ................................ 155

6.1 Access Mediation ...................................... 155
  6.1.1 Discretionary Access Control ....................... 155
  6.1.2 Mandatory Access Control ........................... 158
  6.1.3 Access Mediation Privileges ........................ 159
  6.1.4 Access Decision Function ........................... 160
  6.1.5 Security Object Management ......................... 162

6.2 Auditing ................................................. 163
  6.2.1 The Audit Server ..................................... 163
  6.2.2 Audit Controls ........................................ 165
  6.2.3 Event Reporting ..................................... 167
  6.2.4 Resource Exhaustion and Error Processing ........ 168
  6.2.5 Analysis Tools and Support Services ............... 171
**Final Evaluation Report Digital Open VMS and SEVMS, Version 6.1, with VAX or Alpha**

**TABLE OF CONTENTS**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.6</td>
<td>Startup and Shutdown</td>
<td>171</td>
</tr>
<tr>
<td>6.2.7</td>
<td>DECnet Auditing</td>
<td>171</td>
</tr>
<tr>
<td>6.3</td>
<td>Identification and Authentication</td>
<td>171</td>
</tr>
<tr>
<td>6.3.1</td>
<td>User Accounts</td>
<td>172</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Account Restrictions</td>
<td>172</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Passwords</td>
<td>172</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Login Processing</td>
<td>173</td>
</tr>
<tr>
<td>6.3.5</td>
<td>Secure Terminal Server</td>
<td>173</td>
</tr>
<tr>
<td>6.3.6</td>
<td>Break-in Controls</td>
<td>173</td>
</tr>
<tr>
<td>6.3.7</td>
<td>Authentication for DECnet Applications</td>
<td>174</td>
</tr>
<tr>
<td>6.4</td>
<td>Object Reuse</td>
<td>175</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Pages of Memory</td>
<td>175</td>
</tr>
<tr>
<td>6.4.2</td>
<td>General Purpose Registers</td>
<td>175</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Vector Processor Registers</td>
<td>176</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Disk Files</td>
<td>176</td>
</tr>
<tr>
<td>6.4.5</td>
<td>Disk and Tape Volumes</td>
<td>176</td>
</tr>
<tr>
<td>6.4.6</td>
<td>Printers</td>
<td>176</td>
</tr>
<tr>
<td>6.5</td>
<td>Assurances</td>
<td>177</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Hardware Testing</td>
<td>177</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Software Testing</td>
<td>178</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Informal Security Policy Model</td>
<td>179</td>
</tr>
<tr>
<td>6.5.4</td>
<td>Rating Maintenance Phase</td>
<td>180</td>
</tr>
</tbody>
</table>

**III TCSEC Evaluation Requirements**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Evaluation as a C2 System</td>
<td>183</td>
</tr>
<tr>
<td>7.1</td>
<td>Discretionary Access Control</td>
<td>185</td>
</tr>
<tr>
<td>7.2</td>
<td>Object Reuse</td>
<td>185</td>
</tr>
<tr>
<td>7.3</td>
<td>Identification and Authentication</td>
<td>186</td>
</tr>
<tr>
<td>7.4</td>
<td>Audit</td>
<td>187</td>
</tr>
<tr>
<td>7.5</td>
<td>System Architecture</td>
<td>188</td>
</tr>
<tr>
<td>7.6</td>
<td>System Integrity</td>
<td>189</td>
</tr>
<tr>
<td>7.7</td>
<td>Security Testing</td>
<td>191</td>
</tr>
<tr>
<td>7.8</td>
<td>Design Specification and Verification</td>
<td>192</td>
</tr>
<tr>
<td>7.9</td>
<td>Security Features User's Guide</td>
<td>192</td>
</tr>
<tr>
<td>7.10</td>
<td>Trusted Facility Manual</td>
<td>193</td>
</tr>
<tr>
<td>7.11</td>
<td>Test Documentation</td>
<td>194</td>
</tr>
<tr>
<td>7.12</td>
<td>Design Documentation</td>
<td>195</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Evaluation as a B1 System</td>
<td>199</td>
</tr>
<tr>
<td>8.1</td>
<td>Discretionary Access Control</td>
<td>199</td>
</tr>
<tr>
<td>8.2</td>
<td>Object Reuse</td>
<td>200</td>
</tr>
<tr>
<td>8.3</td>
<td>Labels</td>
<td>201</td>
</tr>
<tr>
<td>8.4</td>
<td>Label Integrity</td>
<td>202</td>
</tr>
<tr>
<td>8.5</td>
<td>Exportation of Labeled Information</td>
<td>203</td>
</tr>
<tr>
<td>8.6</td>
<td>Exportation to Multilevel Devices</td>
<td>204</td>
</tr>
<tr>
<td>8.7</td>
<td>Exportation to Single-Level Devices</td>
<td>204</td>
</tr>
</tbody>
</table>

**FINAL: 24 October 1996**
8.8 Labeling Human-Readable Output .................................................. 205
8.9 Subject Sensitivity Labels .............................................................. 206
8.10 Device Labels ............................................................................. 207
8.11 Mandatory Access Control ........................................................... 207
8.12 Identification and Authentication .................................................. 208
8.13 Audit ............................................................................................ 209
8.14 System Architecture .................................................................... 210
8.15 System Integrity ........................................................................... 212
8.16 Security Testing ........................................................................... 213
8.17 Design Specification and Verification .......................................... 214
8.18 Security Features User’s Guide ...................................................... 214
8.19 Trusted Facility Manual ................................................................. 215
8.20 Test Documentation ..................................................................... 216
8.21 Design Documentation ................................................................ 217

9 Evaluator’s Comments ................................................................... 219
9.1 Other OpenVMS/SEVMS Security Features ............................... 219
  9.1.1 Password Management Options ............................................... 219
  9.1.2 Break-in Detection .................................................................. 219
  9.1.3 Extended Discretionary Access Controls ................................. 219
  9.1.4 Application Subsystems ........................................................... 220
9.2 General Comments ...................................................................... 220
  9.2.1 VMSCluster ............................................................................ 220
  9.2.2 Internal Consistency Checks ...................................................... 221
  9.2.3 Audit File Switching ................................................................. 221
  9.2.4 Audit Alarms ........................................................................... 221
  9.2.5 DECwindows ........................................................................ 221
  9.2.6 Trusted Path ........................................................................... 221
  9.2.7 Security Enhancement Services ............................................... 222
  9.2.8 Classified Mail ........................................................................ 222
  9.2.9 Integrity Policy ........................................................................ 222
  9.2.10 Classification of Files and Directories .................................... 222

IV Appendixes .................................................................................. 225
A Evaluated Hardware List (VAX and Alpha) ................................. 227
  A.1 Processors ................................................................................ 227
  A.2 Processors - Grouped by System Bus .......................................... 229
  A.3 I/O Busses for AlphaServer 2100 Models 4/200, 4/275 ............... 231
  A.4 I/O Bus Adapters and Controllers ............................................ 231
  A.5 Disk Devices ............................................................................. 234
  A.6 Tape Devices ............................................................................. 236
  A.7 Terminals .................................................................................. 236
  A.8 Printers ...................................................................................... 238
  A.9 Miscellaneous .......................................................................... 239

B Evaluated Software List (VAX and Alpha) .................................... 241
### FIGURES

3.1 Processor Status Longword (VAX) .................................................. 16
3.2 Virtual Address Space (VAX) .......................................................... 27
3.3 Virtual Address Format (VAX) .......................................................... 28
3.4 Page Table Entry Format (VAX) ......................................................... 28
3.5 System Virtual to Physical Address Translation (VAX) ....................... 31
3.6 P0 Virtual to Physical Address Translation (VAX) ................................ 33
3.7 Alpha Instruction Formats ................................................................. 36
3.8 Processor Status Register (Alpha) ..................................................... 39
3.9 Virtual Address Regions (Alpha) ....................................................... 44
3.10 Virtual Address Format (Alpha) ....................................................... 45
3.11 Page Table Entry (Alpha) ............................................................... 45
3.12 Virtual Address Translation (Alpha) ................................................ 47
3.13 VAX-11/780 System Configuration .................................................... 49
3.14 VAX 8200 System Configuration ....................................................... 50
3.15 DEC 7000 System Configuration ....................................................... 51
3.16 A single architecture CI VMScluster ............................................... 59
3.17 A mixed architecture CI VMScluster ................................................ 60
3.18 A single architecture MI VMScluster ................................................ 65
3.19 A mixed architecture MI VMScluster ................................................ 66
3.20 Basic Ethernet Connection ............................................................... 67
3.21 DELNI Connection ...................................................................... 69
3.22 A single architecture MI VMScluster ............................................... 70
3.23 A mixed architecture MI VMScluster ............................................... 71

4.1 System Communication Architecture Block Diagram for VAX................. 84
4.2 Invalid Global Page PTE (VAX) ........................................................ 89
4.3 VAX S0 Space .............................................................................. 93
4.4 VAX P1 Space .............................................................................. 94
4.5 VAX P0 Space .............................................................................. 95
4.6 Alpha System Space ..................................................................... 96
4.7 Alpha P1 Space ............................................................................ 97
4.8 Lock Management Data Structures .................................................. 108
4.9 Distributed Lock Management ....................................................... 110
4.10 I/O Data Structures ................................................................. 122
4.11 Read I/O Request with Volume Shadowing ..................................... 124
4.12 Write I/O Request with Volume Shadowing ..................................... 125

5.1 The Access Rights Block ................................................................. 136
5.2 LOGIN Privilege Processing ............................................................. 140
5.3 Image Activation Privilege Processing ............................................... 141
5.4 Image Rundown Privilege Processing .............................................. 141
5.5 $SETPRV Privilege Processing (Working Mask) .................................. 142
5.6 The Objects Rights Block ............................................................... 144
FIGURES

6.1 Functional Flow of Audit Mechanism .................................................. 164
# TABLES

3.1 Internal Processor Registers (VAX) .............................................. 15  
3.2 PSL Description (VAX) ............................................................... 17  
3.3 Page Table Entry Protection Codes (VAX) ..................................... 29  
3.4 PS Description (Alpha) .............................................................. 39  
3.5 Internal Processor Registers (Alpha) ........................................... 40  
3.6 Page Table Entries (Alpha) ....................................................... 46  

5.1 OpenVMS/Security Enhanced VMS (SEVMS) Non-Security- Relevant Privileges .......................... 138  
5.2 OpenVMS/SEVMS Security-Relevant Privileges ................................ 139
This page intentionally left blank
EXECUTIVE SUMMARY

The security protection provided by Digital Equipment Corporation’s OpenVMS and SEVMS operating systems has been examined by the National Security Agency (NSA). The security features of the systems were examined against the requirements specified by the DoD Trusted Computer System Evaluation Criteria (TCSEC)[2], dated December 1985, and applicable TCSEC interpretations, in order to establish a rating for each system.

It has been determined that the highest class at which OpenVMS satisfies all the specified requirements of the TCSEC is class C2, and the highest class for SEVMS is class B1.

The OpenVMS and SEVMS operating systems are multiuser, general purpose time-sharing systems that combine a 32-bit architecture (VAX platforms) or 64-bit architecture (Alpha platforms), and a virtual memory operating system.

The Virtual Address eXtension (VAX) hardware family includes the VAX 700 series, VAX 4000 series, VAX 6000 series, VAX 7000 series, VAX 8000 series, VAX 9000 series, VAX 10000 series, and the MicroVAX, VAXserver, and VAXstation processors.

The Alpha processors include the AlphaServer 2100, DEC 7000, and DEC 10000.

There are four hardware configurations defined for evaluated VAX systems. These are (1) a standalone system, (2) a Computer Interconnect (CI) VMScluster, (3) a Local Area Network Interconnect (NI) VMScluster, and (4) a Mixed Interconnect (MI) VMScluster, combining configurations (2) and (3). The same is true for evaluated Alpha systems. In addition, it is possible to construct VMScluster systems that consist of VAX and Alpha processors in configurations (2), (3), and (4). These VMScluster systems are known as Mixed Architecture VMScluster (MAVC) systems.

OpenVMS and SEVMS protected objects include disk and tape files, volumes, devices, and other system resources that can be shared among users.

C2 OpenVMS (VAX and Alpha)

A system that has been rated as a class C2 system provides a Trusted Computing Base (TCB) that enforces a Discretionary Access Control (DAC) policy, making users individually accountable for their actions through logon procedures, auditing of security-relevant events, and isolation of protected resources.

A user can access OpenVMS by providing a unique valid username and password pair. User accounts are maintained by a system administrator in a User Authorization File (UAF). Each record within the UAF includes the privileges that define the role of the user.

OpenVMS provides two related mechanisms to control the access that users have to protected objects. These are protection based on access control lists and protection based on user category (system, group, owner, and world).

There are three methods used within OpenVMS to specify and control audit data: (1) administrative audit commands, (2) audit entries in access control lists, and (3) user records in the UAF. Audit data collected in the audit log can be transformed into human-readable form by means of the ANALYZE/AUDIT utility. The ability to do selected auditing, and the ability to perform audit log filtering, are present in OpenVMS.

B1 SEVMS (VAX and Alpha)
EXECUTIVE SUMMARY

A system that has been rated as a class B1 system provides all of the features required for a C2 system. In addition, an informal statement of the security policy model, data sensitivity labeling, and Mandatory Access Control (MAC) over subjects and objects must be present. There must also be a capability to accurately label exported information.

SEVMS provides all of the security features of OpenVMS and enforces a MAC policy. Contained within the UAF for each SEVMS user is a maximum clearance, a minimum clearance, and a default clearance. If a user specifies a clearance during login, it must be within the range specified in the UAF. If no clearance is specified, the user’s default is used.

SEVMS implements MAC labels on objects as a pair of classification blocks (maximum and minimum), with each classification block containing a secrecy level and secrecy categories. An access decision algorithm, based on the Bell-LaPadula Model, is used by the TCB to determine if a subject is allowed access to an object. SEVMS includes classification labels in audit records and provides human-readable labels on printed output.
Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha

Part I

System Overview
Chapter 1

Introduction

This report gives evidence and analysis of the security features and assurances provided by the Digital Equipment Corporation OpenVMS VAX Version 6.1, OpenVMS Alpha Version 6.1 with ALPRAMP01.061, SEVMS VAX Version 6.1, and SEVMS Alpha Version 6.1 with ALPRAMP01.061 operating systems as well as for mixed VMScluster systems involving OpenVMS Version 6.1 running on Virtual Address eXtension (VAX) and Alpha platforms\(^1\) or SEVMS Version 6.1 running on VAX and Alpha platforms\(^2\).

OpenVMS VAX Version 6.1 and OpenVMS Alpha Version 6.1 with ALPRAMP01.061 were analyzed relative to the C2 DoD Trusted Computer System Evaluation Criteria\(^2\) (TCSEC) requirements and applicable interpretations, and SEVMS VAX Version 6.1 and SEVMS Alpha Version 6.1 with ALPRAMP01.061 were analyzed relative to the B1 TCSEC requirements and applicable interpretations.

1.1 Evaluation Process Overview

In January, 1981, the United States Department of Defense (DoD) Computer Security Center was formed under the direction of the National Security Agency (NSA). The DoD Computer Security Center was tasked with encouraging the widespread availability of trusted computer systems that could be used by facilities processing classified or other sensitive information. The DoD Computer Security Center’s charter, contained within DoD Directive 5215.1, Computer Security Evaluation Center, dated 25 October 1982, prescribes that the Center establish and maintain “...technical standards and criteria for the security evaluation of trusted computer systems that can be incorporated readily into the Department of Defense component life-cycle management process...” In August, 1985, the name of the DoD Computer Security Center was changed to the National Computer Security Center (NCSC). Today, the NCSC is supported via NSA’s Trusted Product Evaluation Program (TPEP) and associated personnel. The TPEP focus is the evaluation of commercial operating systems, enclosed networks, network components, and database systems.

In August, 1983, the DoD Computer Security Center issued the Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-001-83. Because of the bright orange color of its covers, the document became known as the “Orange Book.” It was also known by the acronym TCSEC and the word “Criteria.” In December, 1985, the “Orange Book” was reissued as Department of Defense Standard 5200.28-STD. Subsequent references to the TCSEC, the “Orange Book,” or to the “Criteria” within this document will imply DOD 5200.28-STD.

Within the Foreword of DoD 5200.28-STD, then Assistant Secretary of Defense Donald C. Latham states that the purpose of the standard is “...to provide technical hardware/firmware/software security criteria and associated technical evaluation methodologies in support of the overall ADP system security policy, evaluation and approval/accreditation responsibilities promulgated by DoD Directive 5200.28.”

\(^1\)The Alpha product must be OpenVMS Alpha Version 6.1 with ALPRAMP01.061.
\(^2\)The Alpha product must be SEVMS Alpha Version 6.1 with ALPRAMP01.061
The TCSEC established specific requirements that a computer system must meet in order to achieve a predefined level of trustworthiness. The TCSEC levels are arranged hierarchically into four main divisions of protection, each with certain security-relevant characteristics. These divisions are D, C, B, and A, with division A being for systems that are the most trustworthy. These divisions are in turn subdivided into hierarchical classes, as follows: Classes C1 and C2 (for Division C); Classes B1, B2, B3 (for Division B), and Class A1 (for Division A). The higher a class notation within a division, the more trustworthy a product belonging to that class is. For example, a B3 product is more trustworthy than is a B1 product.

In addition to the TCSEC, there are two “interpretations” of the TCSEC that have been issued by the NCSC. These are the National Computer Security Center Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, NCSC-TG-005, dated July, 1987, having TNI as its acronym, and the National Computer Security Center Trusted Database Management System Interpretation of the Trusted Computer System Evaluation Criteria, NCSC-TG-021, dated April, 1991, and having TDI as its acronym. When a product is evaluated under the TPEP, it is evaluated against the TCSEC or one of its interpretations, namely the TNI or the TDI.

Today, there are three phases associated with the TPEP, these being the Pre-Evaluation Phase, the Evaluation Phase, and the Rating Maintenance Phase (RAMP).

During the Pre-Evaluation Phase, NSA first reviews the vendor’s product proposal to determine if the product is a candidate for evaluation. If so, a technical assessment of the product is performed by NSA, with the outcome being (a) a recommendation to schedule an Intensive Preliminary Technical Review (IPTR) for the product; or (b) a recommendation that the vendor receive commercial or NSA advice relating to the product so that the product can eventually be subjected to an IPTR; or (c) a recommendation to terminate the evaluation effort because of technical shortcomings of the product. The purpose of the IPTR is to accurately assess completeness of both the product and the evaluation evidence, and to ensure the vendor’s readiness to enter the Evaluation Phase.

After TPEP Management declares that the vendor’s product is ready for evaluation, the Evaluation Phase is entered, with NSA providing an evaluation team (ET) to perform the evaluation. The vendor then provides the ET with system level, developer-oriented training for the product. Training is followed by a comprehensive review of the system design by the ET. The ET performs security analysis of the product design, including hardware and software components of the system, and reviews required documentation such as that relating to security testing, and to the process the vendor will implement when the Rating Maintenance Phase is entered. This later document is the Rating Maintenance Plan (RM-Plan), which is the vendor document that describes the mechanisms, procedures, and tools that the vendor will use to meet the RAMP Requirements. The ET uses the information gathered during the design analysis to write an Initial Product Assessment Report (IPAR) which is then presented to a Technical Review Board (TRB). During a meeting of the TRB, the ET defends the IPAR and provides the TRB with details about the approach the ET will take while performing security testing on the product. After the ET performs security testing on the product at a site provided by the vendor, the results are included in the IPAR which is then transformed into a Final Evaluation Report (FER). The ET presents the results of their security testing at a meeting of the TRB, known as the “Final TRB,” and the TRB makes a recommendation to TPEP Management, which then makes the final decision as to entry on the Evaluated Products List (EPL).

The Rating Maintenance Phase, which is described within Rating Maintenance Phase Program Document, NCSC-TG 013, Version-2, National Computer Security Center, 1 March 1995, is entered when the EPL entry is made for the product. RAMP provides a mechanism for a vendor to maintain the TCSEC rating initially awarded to the product throughout the product’s life cycle, and the vendor performs the majority of the work, such as security analysis, during RAMP. The vendor, through a Vendor Security An-
alyist (VSA), works with the NSA Technical Point of Contact (TPOC) to discuss proposed changes to the product. In the event changes are major, NSA might assign a Security Analysis Team (SA-Team) for the RAMP effort, with the TPOC and the VSA serving as SA-Team members with, perhaps, other VSA and/or NSA evaluators. The composition of an SA-Team is recommended to TPEP Management by a Future Change Review Board (FCRB), which is a panel that reviews future evaluated product changes and makes applicable recommendations to TPEP Management. When changes to the product have been completed, the vendor prepares a Rating Maintenance Report (RMR), which is a summary identifying each change that has been made to the product since the previous evaluated release of the product. The RMR is provided to TRB members and a “RAMP TRB” is conducted. The TRB then makes a recommendation to TPEP Management, which then makes the final decision as to entry on the EPL. A RAMP Cycle is the period of time between the dates of two consecutive EPL entries for a product, and a product can be involved in many RAMP Cycles.

1.2 Background and History

Digital Equipment Corporation, based in Maynard, Massachusetts, was founded in 1957 by Kenneth Olsen. In April 1970, Digital introduced the PDP-11/20 minicomputer, which was based on a 16-bit architecture. In October 1977, the first VAX computer, the VAX-11/780, was introduced. The VAX-11/780, like all of its successors, supports the OpenVMS/Security Enhanced VMS (SEVMS) operating system.

The VAX multiuser, general purpose computer system combines a 32-bit architecture and a virtual memory system to provide essentially unlimited program space. Included in the VAX family are the VAX 700 series, VAX 4000 series, VAX 6000 series, VAX 7000 series, VAX 8000 series, VAX 9000 series, VAX 10000 series, MicroVAX, VAXserver, and VAXstation processors.

In 1986, VMS Version 4.3, running on selected VAX platforms, was evaluated by the National Computer Security Center (NCSC) at the C2 level of trust against the DoD Trusted Computer System Evaluation Criteria[1], dated 15 August 1983. The product was assigned Evaluated Products List (EPL) entry CSC-EPL-86/004.

In 1987, Digital offered SEVMS Version 4.4 as an unevaluated TCSEC product. This product had much of the functionality of VMS Version 4.3, and also provided Mandatory Access Control (MAC) features.

In 1993, Digital changed the name of its VMS operating system to OpenVMS, and that same year OpenVMS VAX Version 6.0 and SEVMS VAX Version 6.0 were evaluated by the NCSC at the C2 and B1 levels of trust, respectively, against the TCSEC, dated December, 1985. OpenVMS VAX Version 6.0 was assigned EPL entry EPL-SUM-93/002, and SEVMS VAX Version 6.0 was assigned EPL entry EPL-SUM-93/003. In 1994, the EPL entries for these two products were changed to reflect the inclusion of mandatory updates. The resulting products, OpenVMS VAX Version 6.0 with VAXMUP03_060, and SEVMS VAX Version 6.0 with SEVMS_VAXMUP03_060, were assigned EPL entries CSC-EPL-93/002 and CSC-EPL-93/003, respectively.

In 1995, OpenVMS VAX Version 6.1 and SEVMS VAX Version 6.1 were evaluated by the NCSC at the C2 and B1 levels of trust, respectively, against the TCSEC, dated December, 1985. OpenVMS VAX Version 6.1 was assigned EPL entry CSC-EPL-95/002.A, and SEVMS VAX Version 6.1 was assigned EPL entry CSC-EPL-95/003.A.

Alpha is a family of Digital processors that combines a 64-bit architecture and a virtual memory system to

---

3 This evaluation was a Rating Maintenance Phase (RAMP) Action.
provide essentially unlimited program space. Included in the Alpha family are the DEC 7000 series, DEC 10000 series, and AlphaServer 2100 systems. In 1993, OpenVMS and SEVMS were ported to the Alpha architecture from the VAX architecture. The focus of the current RAMP Action is the port of OpenVMS and SEVMS to Alpha, specifically the products OpenVMS Alpha V6.1 and SEVMS Alpha V6.1, as well as mixed-architecture clusters. These mixed-architecture clusters involve OpenVMS running on VAX and Alpha platforms or SEVMS running on VAX and Alpha platforms.

1.3 Document Organization

This report consists of four major parts. The first part contains four chapters that discuss the implementation of the system in detail. Chapter 1 is an introduction. Chapter 2 provides an overview of the system hardware and software architecture. Chapter 3 describes the VAX and Alpha hardware and the Digital VMScluster architecture. Chapter 4 discusses the functionality of the primary system software components.

The second part contains two chapters that discuss the security functionality of the system in more detail. Chapter 5 defines the subjects and objects that are controlled by the OpenVMS/SEVMS Trusted Computing Base (TCB), and Chapter 6 describes the TCB protection mechanisms.

The third part contains two chapters that discuss specifically how each system addresses the applicable TCSEC requirements. Chapter 7 provides a mapping between the C2 requirements specified in the TCSEC and the OpenVMS features that fulfill those requirements. Chapter 8 provides a mapping between the B1 requirements and the SEVMS features that fulfill those requirements. Within chapters 7 and 8, additional requirements at higher evaluation classes that are met by OpenVMS/SEVMS are also discussed. Chapter 9 presents evaluator comments on features of the system that may be of interest, but which do not address specific TCSEC requirements at the level of the evaluation.

The fourth part contains five appendixes. Appendix A identifies the specific hardware to which this evaluation applies. Appendix B identifies the specific software. Appendix C provides lists of the executable files in the TCB. Appendix D provides a list of acronyms used throughout the report. Appendix E provides the list of references used to write this report.

1.4 Conventions

Several stylistic conventions were used throughout the report to simplify the language used in the discussions. The term OpenVMS is used to refer to the base C2 operating system. The term SEVMS refers to the B1 Security Enhanced version of the system. Whenever reference is made to generic features of both systems, either the term Open VMS/SEVMS is used or both systems are specifically mentioned. In addition, both VAX and Alpha are implied at all times unless noted otherwise. For example, the use of term OpenVMS/SEVMS within a discussion implies that the discussion applies to VAX as well as Alpha.

The use of acronyms within the report follows the convention that each acronym is expanded the first time it is used within a chapter. The complete list of acronyms is included in Appendix D.

All keywords and filenames relating to OpenVMS and SEVMS are capitalized within the text. The names of system services (see page 79, “System Services”) are capitalized and preceded by a dollar sign ($).
The initial FER for OpenVMS VAX and SEVMS VAX, Version 6.0, and the FER update for OpenVMS VAX and SEVMS VAX, Version 6.1, have contained the term VAXcluster. That term is not used within this FER update; rather, VMScluster is used.
Chapter 2

TCB Overview

OpenVMS/Security Enhanced VMS (SEVMS) is a general purpose, multiuser operating system that runs on VAX, MicroVAX, VAXstation, VAXserver, Alpha, and AlphaServer series computers. OpenVMS/SEVMS includes integrated distributed computing and multiprocessing capabilities. Although a windowing system is included in the operating system for workstations, it is not part of the evaluated configuration. The configurations for OpenVMS/SEVMS include systems from single-user workstations to large clusters of mainframe-class machines supporting thousands of users.

The OpenVMS/SEVMS Trusted Computing Base (TCB) consists of those components of the system that are responsible for all security-related activities and for correctly enforcing the system security policy. A security policy defines the interaction between subjects and objects of the system. Subjects in OpenVMS/SEVMS are processes. A process acts on behalf of a user. Objects in OpenVMS/SEVMS consist of data containers whose contents can be read and/or written by multiple users, and to which a user interface is provided (see page 135, “TCB Protected Resources”). Specifically these objects are as follows:1

- Files and Directories
- Volumes
- Devices
- Queues
- Global Sections
- Shared Logical Name Tables
- Resource Domains
- Common Event Flag Clusters

The security policy for OpenVMS includes enforcement of Discretionary Access Control (DAC) and user authentication policies. The security policy for SEVMS builds on the OpenVMS policy with additions for Mandatory Access Control (MAC) policy enforcement and data sensitivity labeling.

The security policy of OpenVMS/SEVMS is defined and enforced for a VMScluster system. A VMScluster system combines one or more processors. Each processor provides virtual address space separation between processes and four execution modes to support TCB isolation. These cooperating processors share a number of resources, including disk and tape devices, resource locks, and batch and print queues. The ability to share disk devices and resource locks permits files to be simultaneously accessed from different nodes by coordinating record-level file locks. Access to the files in the security database is coordinated in this manner to provide consistent security information throughout the VMScluster.

A VMScluster system may be configured as either a common environment cluster or as a mixed environment cluster. A common environment cluster uses a single copy of the security database. These files are shared by every node. A mixed environment cluster permits each node to have a separate security database. The

1 Two features of OpenVMS/SEVMS are protected by discretionary security mechanisms although they are not objects. These features are processor capability and security classes.
CHAPTER 2. TCB OVERVIEW

Trusted Facility Manual (TFM) warns the administrators to use a common environment cluster to conform to the evaluated configuration. This means that all nodes in the VMScluster use the same security database resulting in a common security domain. Several process attributes are defined with the explicit purpose of identifying the node on which the process is running. Other process attributes are entirely independent of the node.

All processors in the VMScluster cooperate to enforce the system security policy. In the B1 configuration, every node must run the evaluated version of SEVMS; in the C2 configuration, every node must run the evaluated version of OpenVMS. The two evaluated configurations are independent (i.e., C2 and B1) and may not be interconnected. The TCB of OpenVMS/SEVMS is composed of software, firmware, and hardware, the combination of which is responsible for enforcing its security policy. The TCB is distributed among all the nodes of a VMScluster.

2.1 Software

The TCB software is made up of images (see page 90, “Images”) and command files. Images are contained in files on disk and are combined in several ways to form executable entities in memory. Some images are loaded directly into the system address space and executed in the protected modes of the hardware. Those images comprise the system executive. Other TCB images are executed in the context of a privileged process. The TCB command files are all executed by privileged processes running the Command Language Interpreter (CLI) image. All files that contain TCB software or data are protected by the DAC mechanisms.

The system executive is composed of approximately 20 major images and a potentially large number of device drivers as described on page 73, “Software Structure”. The system executive supports varied hardware configurations and some system features by conditionally loading images. As an example, device drivers are not loaded if the peripheral is not present in the configuration. The logic that decides what modules of the executive need to be present is discussed on page 130, “System Startup”.

There are several ways that an image may be executed with privilege. An image may execute in a privileged process created by the system specifically to execute that image. An image may also execute in an administrator’s process and utilize the administrator’s privilege. The final way an image may gain privilege is for the image to be administratively designated to run with privilege no matter who runs it. The process of designating such an image is called “installing the image with privilege.” After an image is installed, it is called a “known image.” For the evaluated configuration, the TFM requires that only specific OpenVMS/SEVMS images be installed with privilege. The installed image feature is described on page 100, “Known Images”.

OpenVMS/SEVMS supports a multiprocessing architecture within the distributed architecture. Each processor independently executes the TCB software. Access of subjects to objects within a node is mediated by software in that node. Access of subjects to objects on other nodes is mediated by software in the subject’s or object’s node depending on the specific access mechanism used. Because the security database used by all nodes is contained in a central location, the result is independent of the node that made the decision. Together, all the nodes present a unified security domain with a shared file system and distributed process management functions.

Because the TCB may be running on a number of different processors, the TCB has internal mechanisms to coordinate security-relevant decisions. These mechanisms are described on page 80, “Internal TCB Communication Mechanisms”. Both shared physical memory and network communication devices are used to facilitate this coordination.
2.2 Firmware

Virtual Address eXtension (VAX) and Alpha nodes and many of the associated peripherals contain firmware. The firmware varies on each particular node and peripheral interface. No firmware may be modified by an unprivileged user. Some firmware requires that front panel switches be set to allow the firmware update. An example of this is the CPU microcode for the VAX 8350. Some peripheral interfaces allow privileged software to update their firmware.

2.3 Hardware

There are conceptually three groupings of system hardware: individual processors, individual nodes, and a complete VMScluster. A VAX processor is a single fetch/execute machine. Each VAX processor independently executes an instruction stream from the operating system and user processes. This means that each CPU independently runs scheduling algorithms, I/O services, etc. A VAX processor executes the VAX architecture instruction set and an Alpha processor executes the Alpha architecture instruction set. Each instruction set provides support for multiple process contexts that provide process separation, and hardware access modes that are hierarchical execution domains used to protect the TCB. User programs execute on VAX and Alpha processors under the control of the operating system.

A VAX node contains one or more VAX processors, memory, I/O busses, and peripherals. Similarly, an Alpha node consists of one or more Alpha processors, memory, I/O busses, and peripherals. If there are multiple processors on a node, the processors share memory and busses. Since I/O controllers are on the shared busses and respond to addresses generated on the busses, the processors also share the local I/O devices. A process (see page 98, “Process Management”) executes only on the processors of a single node during its existence. A second type of node is the Hierarchical Storage Controller (HSC) node. An HSC node is an intelligent disk and tape controller (see page 58, “HSC”). Disk and tape peripherals may be connected to either an HSC or VAX node or Alpha node. Other types of peripherals may be installed only in VAX or Alpha nodes.

A VMScluster system is a collection of nodes and the hardware that connects them. Nodes in a VMScluster are connected by high speed communications channels. These channels are used by TCB software to distribute the file system and some process management functions over the entire VMScluster.

The following chapters discuss the hardware from the perspective of a processor, node, and VMScluster.
CHAPTER 2. TCB OVERVIEW

This page intentionally left blank
Chapter 3

Hardware Architecture

This chapter describes the hardware for both OpenVMS and Security Enhanced VMS (SEVMS). The first section describes the characteristics of a single VAX processor. The construct of the second section is similar to the first, with the focus being the Alpha processor. The third section describes the hardware devices that can be configured into a complete VAX or Alpha system that can be either a stand-alone OpenVMS/SEVMS system or a VMScluster node. The final section describes the additional hardware components that can be part of a VMScluster configuration.

3.1 VAX Architecture

This section describes basic concepts and capabilities of the Virtual Address eXtension (VAX) architecture. An overview of the processor is presented followed by discussions of the instruction set, process management, interrupts and exceptions, memory management, and I/O management.

3.1.1 Architecture Overview

The VAX architecture, which is an example of Complex Instruction Set Computing (CISC), is designed to support a highly flexible, multiprogramming environment. Multiprogramming is supported through context switching, process isolation, virtual addressing, memory management, and hardware access modes that support hierarchical execution domains.

There are four hardware access modes. They are, ranging from most to least privileged: kernel, executive, supervisor, and user. Privileged instructions can only be executed in kernel mode. The memory is protected using protection codes for each memory page. For each access mode, a page may be readable, readable and writable, or inaccessible. If a page is accessible in a particular access mode, it is also accessible in the modes that are more privileged than that particular mode. For example, if a page is readable in supervisor mode, it is also readable in executive and kernel modes.

The following is true of all of the VAX processors in the evaluated configuration. They all provide a 32-bit virtual address space (4 Gbytes), they all implement the same base instruction set (however, certain instructions are emulated in software on some processors), they all support the four hardware access modes, and they all provide 16 32-bit general purpose registers, 12 addressing modes, and 32 interrupt levels. In addition, each processor also supports programmable clocks and the I/O subsystem. For all of the VAX processors, physical memory is divided into two distinct address spaces. The first is addressable physical memory, termed “Memory space.” OpenVMS/SEVMS and application programs run in Memory space. The second address space is used to provide addressability to device I/O registers for most of the processors examined. This is called I/O space. The physical entity that is actually addressed is not a memory board, but is often a bus adapter. The I/O interface to devices is implemented through read and write operations.
CHAPTER 3. HARDWARE ARCHITECTURE

on the registers contained in I/O space. Both data and commands are read from and stored in I/O space. All of physical memory, including that used to support I/O devices, is addressable. Other processors use a “mailbox I/O” mechanism described on page 32, “Mailbox I/O”.

Each feature of a processor, and its role in the operation of OpenVMS/SEVMS, is described in more detail in the following sections.

3.1.2 Instruction Set

This section describes the instruction set for the VAX architecture. The VAX datatypes and the standard, privileged, subset, and superset instructions are discussed along with the hardware registers.

3.1.2.1 Hardware Registers

As mentioned previously, 16 32-bit general purpose registers, referred to as R0-R15, are provided by a VAX processor. The instruction set provides access to all of the registers. However, four of them have special uses. Those registers contain the program counter (R15), stack pointer (R14), current stack frame pointer (R13), and the argument pointer (R12). The program counter contains the address of the next instruction to be executed. The stack pointer is the address of a processor-defined stack (one of five stacks, four of which are associated with a hardware access mode; the fifth is the interrupt stack (see page 23, “Interrupts and Exceptions”)). The current frame pointer is the address of a data structure, called a stack frame, built for a process when a procedure call is encountered in the executing program; and the argument pointer contains the base address of a list containing procedure parameters. Procedure parameters are contained in the stack corresponding to the mode of the caller. Of the four registers above, only the program counter is inaccessible directly by unprivileged instructions, and the other three registers can be used for other purposes.

3.1.2.2 Internal Processor Registers

There are also several internal processor registers which are Central Processing Unit (CPU) control and status registers that hold memory management or status information or point to stacks corresponding to the hardware access mode of the process. Most of the internal processor registers are loaded during a context switch when a process is scheduled to run. Table 3.1 lists the internal processor registers of interest (it is not a complete list).

The P0 and P1 length and base registers demark the process address space regions (see page 27, “Virtual Address Space”). The system base register and system length register define system space (called S0 space). The system base register contains the physical starting address of the system page table. The P0 base register contains the virtual address of the beginning of the P0 page table. The P1 base register points to the beginning of the unallocated portion of P1 space. The process control block base register points to the beginning of the Process Control Block (PCB) (see page 22, “Process Context”). The system control block base register points to the beginning of the System Control Block (SCB) (see page 24, “System Control Block”). The Interrupt Priority Level (IPL) register specifies the currently executing priority level of the process. A write to this register also loads the IPL field in the Processor Status Longword (PSL). A read from this register actually causes a read from the IPL field in the PSL. The PSL is discussed in the following paragraphs. The Asynchronous System Trap (AST) level register is used to contain the access mode of the
most privileged access mode for which an AST is pending. ASTs are discussed in more detail on page 20, “Asynchronous System Trap (AST)”. The AST level register is set by software. The four console terminal registers are internal processor registers for some consoles. When the registers are present, they are used for all data transfers to and from the console terminal. Consoles are discussed in more detail on page 47, “VAX or Alpha System Node”. The memory management enable register is set when memory mapping is turned on.

The internal processor registers are directly accessible only through two privileged instructions: Move To Privileged Register (MTPR) and Move From Privileged Register (MFPR). The load and store context instructions, LDPCTX and SVPCTX, respectively, indirectly access the internal registers. LDPCTX and SVPCTX are also privileged instructions.

### 3.1.2.3 Program Status Longword (PSL)

In addition to the internal registers, the CPU status is kept in the PSL, a 32-bit entity which is a word of privileged processor status concatenated with the Processor Status Word (PSW). The PSL is shown in Figure 3.1. The PSW, which comprises bits 0:15 of the PSL (numbering from right to left in Figure 3.1) contains the condition codes that indicate the results of the previously executed instruction and the bits that control the action of the processor when exceptions occur. Currently, only bits 0:7 are used. Bits 8:15 are always zeroed.

The rest of the PSL contains the privileged processor status and can be modified only by the Return from Exception or Interrupt (REI) and MTPR instructions. A complete description of the fields in the PSL is shown in Table 3.2. The CUR.MOD and PRV.MOD fields indicate the current and previous hardware access modes of the currently executing process, respectively. As indicated in Table 3.2, the PRV.MOD field is
used to store the CUR_MOD field while exceptions are being processed or when change mode instructions are being executed. The PRV_MOD bits are cleared when an interrupt occurs and are restored by the REI instruction. The Interrupt Stack (IS) bit is set to indicate that the process is executing on the interrupt stack. When the IS bit is clear, the process is executing on the process stack indicated by the CUR_MOD field. The First Part Done (FPD) bit is used when an instruction is interrupted in the middle of its execution and needs to be restarted at a point in its operation dependent on the implementation of the instruction and whether the instruction is being emulated. The Compatibility Mode (CM) bit is used to indicate the processor is executing PDP-11 instructions. When the bit is clear, the processor is in “native” mode, or executing the VAX instruction set. Some processors do not execute PDP-11 instructions; the CM bit is clear in that case. However, some PDP-11 instructions are emulated through software. Compatibility mode is entered via execution of the REI instruction with the CM bit in the PSL set on the stack. Native mode is reentered through an exception or interrupt.

PDP-11 instructions are 16-bit instructions. In compatibility mode, the 16-bit address is extended to 32-bits. However, a process that executes in compatibility mode is constrained to execute within the first 64K of its virtual address space. Compatibility instructions can be executed only in user mode. The instruction set is limited and does not include privileged instructions or access to internal processor registers. The first seven general purpose registers of the PDP-11 correspond to the first 16 bits of the first seven registers (registers R0-R6) on the VAX. The program counter, kept in R7 on the PDP-11, corresponds to bits 0:15 of the program counter on the VAX, kept in R15. The upper halves of registers R0-R6 and registers R8-R14 on the VAX are ignored while executing in compatibility mode. The upper half of the program counter is zeroed.
### Table 3.2. PSL Description (VAX)

<table>
<thead>
<tr>
<th>Bit Extent</th>
<th>Name</th>
<th>Mnemonic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Compatibility Mode</td>
<td>CM</td>
<td>When clear, the processor is in VAX mode. When set, the processor is in PDP-11 mode.</td>
</tr>
<tr>
<td>30</td>
<td>Trace Pending</td>
<td>TP</td>
<td>Reserved by Digital; must be zero.</td>
</tr>
<tr>
<td>29 to 28</td>
<td>Reserved</td>
<td></td>
<td>Reserved by Digital; must be zero.</td>
</tr>
<tr>
<td>27</td>
<td>First Part Done</td>
<td>FPD</td>
<td>When set, execution of the instruction addressed by PC cannot simply be started at the beginning, and must be restarted at some other implementation dependent point in its operation.</td>
</tr>
<tr>
<td>26</td>
<td>Interrupt Stack</td>
<td>IS</td>
<td>When set, the processor is executing on the interrupt stack. If clear, the SP of the mode is used.</td>
</tr>
<tr>
<td>25 to 24</td>
<td>Current Access Mode</td>
<td>CUR_MOD</td>
<td>The access mode of the currently executing process.</td>
</tr>
<tr>
<td>23 to 22</td>
<td>Previous Access Mode</td>
<td>PRV_MOD</td>
<td>The Previous Mode is loaded by exceptions and change mode (CHMx) instructions. Restored by REI.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
<td></td>
<td>Reserved by Digital; must be zero.</td>
</tr>
<tr>
<td>20 to 16</td>
<td>Interrupt Priority Level</td>
<td>IPL</td>
<td>The current processor priority.</td>
</tr>
<tr>
<td>15 to 8</td>
<td>Reserved</td>
<td></td>
<td>Reserved by Digital; must be zero.</td>
</tr>
<tr>
<td>7</td>
<td>Decimal Overflow</td>
<td>DV</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Floating Underflow</td>
<td>FU</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Integer Overflow</td>
<td>IV</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Trace Enable</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Negative</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Zero</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Overflow</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Carry</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>
3.1.2.4 Standard VAX Instructions and Data Types

VAX instructions are of variable length and can begin on any byte boundary. The opcode of an instruction can be one or two bytes long. The number of operands, called operand specifiers, can range from zero to six. An operand specifier can be one to 17 bits long depending on the addressing mode and data type.

There are two broad categories of register addressing modes: general and branch. Within the general category, there are two types of addressing: general register and program counter. There are nine general register addressing modes.

The program counter may be addressed as a register using eight different modes. In branch addressing mode, the contents of the program counter are affected. The byte or word displacement is sign-extended to 32 bits and added to the contents of the program counter.

Instructions are processed first by evaluating the type of access (read, write, modify, address, branch, and variable-length bit field) and the address of each operand in the order it occurs within the instruction. The operation indicated by the opcode is then performed. The results are then stored, again according to the order and type of operand specifiers within the instruction.

There are six data type groups, such as character-string and integer, that are manipulated by the instruction set. An integer can be contained in a byte (8 bits), a word (16 bits), a longword (32 bits), a quadword (64 bits), or an octaword (128 bits). There is also a variable-length queue entry data type that is minimally two longwords. A queue is composed of queue entries, possibly of varying sizes. A queue may contain zero to 2 billion entries.

The VAX instructions are grouped into following categories: address, arithmetic, character string, control, cyclic redundancy check, decimal string, edit, floating point, index, integer, logic, multiple register, privileged, procedure call, processor status longword, queue, and variable-length bit field. Generally, the operation of the instruction groups is implied by their names.

All programs, regardless of the language in which they are coded, implement a standard VAX procedure call interface. Procedure call instructions save the registers used by the called procedure, (including the program counter, stack, frame, and argument pointers), pass an argument list to the called procedure, and set the arithmetic trap enable bits in the PSW to a specific state. When a procedure call instruction is encountered, the argument list and the saved registers are stored in the process stack corresponding to the hardware access mode of the process. The argument list and registers form part of a stack frame that also includes a register save mask, several control bits, and a word used to point to a condition handler, if needed. The condition handler field is cleared during execution of a procedure call. At the end of a procedure call instruction, the frame pointer contains the address of the frame containing the saved information. The return instruction uses the frame pointer to restore the state of the process after the called procedure has been executed.

The control instructions either modify or check the condition codes and may branch to a specified address depending upon the results. The edit instruction converts a packed decimal string to a character string for output. The instructions that manipulate the PSL allow the setting and clearing of trap enable bits and the reading of the contents of the PSL into a destination address.
3.1.2.5 Interlocking and Atomic Operation Instructions

There are seven interlocking instructions included in the VAX instruction set. They are used to control multiprocessor and I/O device access to shared variables. The instructions are: branch-on-bit-set-interlocked, branch-on-bit-clear-interlocked, add-aligned-word-interlocked, insert-queue-head-interlocked, insert-queue-tail-interlocked, remove-queue-head-interlocked, and remove-queue-tail-interlocked. Only other interlocked instructions are locked out during execution of one of the interlocked instructions.

Atomic operation instructions are those that execute entirely before another processor attempts to modify the same location as the first instruction. All instructions that write to a longword (32 bits), longword-aligned location are guaranteed by the hardware not to have any partial results overwritten by another processor. A move to longword instruction (MOVLI) is an example of an instruction used to perform an atomic operation. For example, a Page Table Entry (PTE) is modified using the MOVLI instructions.

3.1.2.6 Privileged Instructions

The privileged instructions provide access to internal processor registers and the capability of changing the security state of the machine. All of the privileged instructions must be executed while in kernel mode. The privileged instructions are: Extended Function Code (XFC), halt (HALT), load process context (LDPCTX), MFPR and MTPR, REI and save process context (SVPCTX). The HALT instruction halts the processor.

The XFC is used to execute non-standard microcode in the writable control store or simulator software that runs in kernel mode. The request is controlled by information stored in the system control block that is explained on page 24, “System Control Block”.

The LDPCTX and SVPCTX instructions are used during context switching as explained later in this section (page 22, “Context Switching”). The MTPR and MFPR instructions allow access to the internal processor registers.

3.1.2.7 Parameter Validation Instructions

There are two instructions, PROBER and PROBEW, that are used by OpenVMS/SEVMS software to verify that a less privileged caller has read or write access to the parameters that are passed to a service routine. The PROBE instructions take three parameters: an access mode, a length, and a base address. Read (PROBER) or write (PROBEW) access to the first and last bytes specified by the length and base address is checked by examining PTEs. The mode used for the protection check is the less privileged of the specified mode and the previous hardware access mode (PRV.MOD) of the PSL (whichever is of the largest numerical value). If the caller does not have access or there is a length violation, an exception is generated.\footnote{The PROBE instructions can also cause a page fault if the referenced page is not in the translation buffer.}

The PROBE instructions are not privileged. They can be executed in any hardware access mode.

3.1.2.8 Change Mode Instructions

The change mode instructions allow a user to explicitly change from one hardware access mode to another, thereby providing access to protected code and data. Change mode instructions can only be issued to progress
to a mode equal to or more privileged than the current mode. They can be issued from any mode. However, if an illegal operation is attempted, such as executing a CHMS (change mode to supervisor) instruction from executive mode, then no change in mode results and an access-control-violation fault occurs. This check is done before any other change mode processing takes place.

Change mode instructions cause exceptions to be generated that are handled by service routines configured to execute for the specified mode change. Arbitrary code cannot be executed by a change mode instruction; an argument to the opcode is required that causes a particular routine that runs in the specified mode to be executed. This is how system services are implemented in OpenVMS/SEVMS (see page 79, “System Services”).

### 3.1.2.9 Subset Instructions

Parts of the VAX architecture may be partitioned out and supplied as options to certain processors, or omitted entirely. The minimum set of instructions is called the kernel set. The kernel set contains 175 of the 304 instructions in the full VAX instruction set. The features of the architecture that are partitioned are the following: floating point instructions, string instructions, compatibility mode instructions, and processor registers. The floating point instructions can be further subset by data type. String instructions include the decimal string, edit, cyclic redundancy check, and character string instructions. Two of the character string instructions are part of the kernel subset and are always included (MOVC3 and MOVC5). All other instructions in the string subset may be omitted on a per-instruction basis.

Subset processors can emulate omitted instructions with the support of OpenVMS/SEVMS. The emulation method depends on the instruction type. For example, string instructions are emulated through the instruction emulation exception. The exception handler then runs to emulate the instruction. In this case, the exception handler runs in the hardware access mode in which the instruction was issued, not necessarily kernel mode.

Compatibility mode and floating point instructions are emulated entirely by software (through exception handlers). A reserved operand fault is generated if compatibility mode is not supported by the processor. A reserved instruction fault is generated for omitted floating point instructions. The processor registers that may be omitted include several interval timer registers, the time-of-year clock register, the four console registers, and the performance monitor enable register.

### 3.1.2.10 Asynchronous System Trap (AST)

ASTs provide a method to notify processes of external events and to initiate certain processing with a minimum delay. ASTs are primarily implemented in software. Hardware support is needed because changes in hardware access modes can result from processing ASTs. An AST has an associated hardware access mode. An AST for a less privileged mode than the current mode cannot interrupt the executing process. As mentioned on page 14, “Internal Processor Registers”, the AST Level register contains the most privileged access mode for which an AST is pending. If the access mode for the new AST is greater than or equal to the contents of the AST Level register, an AST interrupt (IPL 2) is raised that causes delivery of the AST to the process.
3.1.2.11 Vector Processing

Some VAX processor architectures (the VAX 6000 and the VAX 9000 series, in particular) support the inclusion of an associated vector processor. The vector processor is used by CPU-intensive, calculation-oriented programs. The instruction set used by the vector processor is unique and consists mainly of arithmetic instructions. The vector processor and CPU share the same memory and data path, and may share the same translation buffer, cache, and internal memory management registers.\(^2\) Instructions executed by the vector processor cannot be executed by a non-vector, or scalar, processor and vice versa. An attempt to execute a vector instruction when a vector processor is not present results in a reserved instruction fault. If present, the vector processor can be enabled or disabled.\(^3\)

The VAX processors that employ the vector processor use an integrated architecture. In the integrated architecture, a scalar processor and its associated vector processor are considered a pair. A processor pair can execute either synchronously or asynchronously. An integrated vector processor can additionally optimize its own execution by overlapping, chaining, or pipelining vector instructions. Vector processor instructions are queued until they can be executed to be sure that no instructions dependent on the results of previous instructions are executed in an improper order.

When a vector instruction is detected by the scalar processor, the instruction is passed to the vector processor including the vector opcode, the scalar source operands, and vector control words. The vector processor loads the necessary data from memory, stores the data, or operates on data previously loaded into the vector processor registers.

Vector processors have 16 vector registers, each containing 64 elements of length 64 bits. The vector registers are accessible by the vector instruction set and are used to contain source and destination operands. There are three vector control registers named the vector length, vector mask, and vector count register, respectively. The control registers are accessible only through the Move From Vector Processor (MFVP) and Move To Vector Processor (MTVP) instructions.\(^4\) There are also five internal vector processor registers, also accessible via the MFVP and MTVP instructions. The most important of these registers is the vector processor status register, which is 32 bits in length and contains information that reflects the current state of the vector processor. The status register indicates whether the vector processor is enabled or busy. Error conditions such as memory fault, illegal opcode, hardware error, or arithmetic exception are also reflected in the status register. Memory management registers and the translation buffer are modified using the MFPR and MTPR instructions that are executed only in kernel mode.\(^5\) The MFVP and MTVP instructions cannot modify the memory management registers or the cache.

Vector instructions have a 2-byte opcode and employ the VAX operand specifies. A vector control word operand that specifies which vector registers are to be used is contained in the instruction. The format of the vector control word operand is instruction dependent. Only vector instructions that reference vector registers contain a vector control word operand. The scalar processor passes the vector control word operand to the vector processor. In addition to indicating which vector registers to use, the control word operand

---
\(^2\) The VAX 6000 processor has its own translation buffer and cache; the VAX 9000 vector processor shares the memory management hardware used by the scalar processor.
\(^3\) Vector processors can become disabled through hardware or software errors or through software control from either the scalar or vector processor. An attempt to execute a vector instruction when a vector processor is disabled results in a vector processor disabled fault.
\(^4\) The MFVP and MTVP instructions are executable in any hardware access mode.
\(^5\) In the VAX 6000, the vector processor contains its own copies of the system page table base and length registers. The firmware for the MFPR and MTPR instructions cause the vector processor copies of these registers to be updated when the system copies are changed.
also enables masking and under/overflow processing (for floating and integer instructions) and specifies the
type of operation to perform for convert and compare instructions.

There are six major classes of vector instructions: memory access, integer, logical and shift, floating-point,
edit, and scalar/vector synchronization. There are approximately 72 vector processor instructions.

There are two classes of vector processor exceptions: memory management and arithmetic. Vector processor
memory management exceptions are handled similarly to VAX processor memory management exceptions.
In addition to the “normal” set of memory management exceptions, a vector processor memory management
exception can be generated from an attempt to access memory in an access mode currently not authorized,
by attempting to reference an out-of-bounds virtual address, by using non-aligned vector operands, or by
a vector instruction that references I/O space (see page 32, “I/O Management”). Arithmetic exceptions
include floating point overflow and underflow and division by zero. The vector processor disables itself when
arithmetic exceptions occur.

When a process using the vector processor is to be context switched, the scalar processor disables the vector
processor (through the MTPR instruction). Once the vector processor is disabled, the vector context of the
process is no longer accessible.

3.1.3 Process Management

This section discusses the definition of a hardware process context and how context switching is performed
on a VAX.

3.1.3.1 Process Context

A process is the entity that is scheduled for execution by the processor. A process has an address space
represented by page tables and both a hardware and software context. From the hardware point of view, a
process context is a set of instructions and data defined by a PCB. The PCB contains images of 14 general
purpose registers (the processor stack pointer is omitted), the PSL, the program counter, the four per-process
stack pointers, the process virtual memory base addresses and lengths, and control fields. The control fields
contain images of a performance-monitor-enable register and an AST level.

3.1.3.2 Context Switching

In OpenVMS/SEVMS, context switching is defined as saving the contents of the privileged registers in the
PCB of the currently executing process and then loading a new context from another PCB. This involves
loading the process control block register and the use of the LDPCTX and SVPCTX instructions. Process
context switching is done in kernel mode only. The SVPCTX instruction uses the process control block
base register to locate the PCB of the currently executing process. The general registers are saved into the
PCB. The program counter and PSL on the top of the current stack are also saved. If the interrupt stack
bit is clear when the save process context instruction is executed, the interrupt stack pointer bit is set (the
scheduler software executes using the interrupt stack) and the current priority level is made 1 (the least
urgent interrupt level) because of the use of the interrupt stack. The reason for this is that it is assumed
that the save and load process context instructions are used as part of interrupt processing (the scheduler
executes at IPL level 3), and that an REI instruction will be used to pass control to the next scheduled process.

The virtual base addresses and the control information may be altered only when the process is executing in kernel mode. When a process is executing, the context is updated in the internal processor registers. When the process is not currently executing, the contents of the registers is stored in the PCB. To alter context information, the process must alter the memory image of the PCB first then move the information to the appropriate internal privileged register. The reason for this is that the context switch instructions (LDPCTX and SVPCTX) do not alter the virtual base address or the control information.

The SVPCTX instruction executes in a slightly different fashion depending upon the processor. The hardware access mode stack pointers are kept in internal registers on some processors. For those processors, the contents of the stack pointer registers are stored in the PCB. Other processors keep only the current hardware access mode stack pointer in an internal register. In that case, the processor alters the contents of the PCB when the current access mode changes.

Practically the reverse of the save operation occurs during execution of the LDPCTX instruction. The PCB address is specified by the process control block base register that is loaded by OpenVMS/SEVMS. The general and virtual base address registers are loaded from the PCB. The translation buffer is cleared (for more information on the translation buffer, see page 28, “Page Tables”). The kernel stack is used for execution. The program counter and the PSL are moved from the PCB to the stack, in anticipation of a subsequent issuance of an REI instruction. As noted in discussing the SVPCTX instruction, the LDPCTX instruction executes differently, depending upon whether or not the processor keeps a copy of all of the stack pointers in internal registers.

3.1.4 Interrupts and Exceptions

Interrupts and exceptions are events that change the flow of processing on a VAX. Interrupts are events that are systemwide in scope and are handled in “system context”. Exceptions are events that affect the executing process and are handled by OpenVMS/SEVMS, usually in the process context.

3.1.4.1 Exceptions

Generally, the form an exception takes is a list of longwords that contains the type of exception that occurred. Exceptions are handled by service routines, the addresses of which are contained in the SCB. There are three categories of exceptions: aborts, faults, and traps. In the VAX architecture, an abort occurs when an instruction leaves the registers and memory in an unpredictable state and the instruction cannot be correctly restarted, completed, simulated, or undone. A fault occurs during instruction execution. The registers and memory are left in a consistent state. A fault is a condition that can be corrected (by software), allowing the instruction to be restarted and to be executed yielding the correct results. A trap is an exception condition that occurs at the end of the executing instruction. The common types of exceptions are: arithmetic, instruction fault, memory management, operand reference, serious system failures, and tracing. Change mode instructions are treated as special exceptions and are discussed on page 19, “Change Mode Instructions”. 

23
3.1.4.2 Interrupts

Interrupts are handled by priority (Interrupt Priority Level (IPL)), as opposed to exceptions, which are handled at the priority in which they occur. If the priority of an interrupt is greater than that of the currently executing process, the processor will raise the priority level to that of the requested interrupt, and dispatch control to the appropriate interrupt handler. When a particular interrupt level is set, all interrupts of lower priority are blocked from service. Interrupts at a particular level can be disabled simply by setting IPL to a level equal to or higher than the level to be disabled. All interrupts are disabled when the IPL is set to 31. The IPL is part of the privileged portion of the PSL and can only be set while in kernel mode through the MTPR instruction.

Interrupts can come from devices, controllers, other processors, or the processor itself. A device interrupt is hardware-generated, as is a processor interrupt. An example of a processor interrupt is the AST Level interrupt which is handled at IPL 2. Kernel mode software can alter the priority of the processor it runs on, but not that of other processors. Kernel mode software also cannot disable interrupts on other processors. Interrupts are processed at the completion of the currently executing instruction or at a pre-defined point during execution where the process state can be contained in the PSL, registers, and the program counter.

Interrupt levels 1 through 15 are reserved for system software. Levels 16 through 23 are reserved for devices, although not every processor has all eight device levels available. There are also eight levels (24-31) reserved for urgent interrupts that include serious system failures and power failure. Not all of the conditions that fall into the urgent category are actually interrupts. One example is a machine check (resulting from events such as bus errors, nonexistent memory or cache parity, translation buffer parity, or control store parity errors) that is treated as an exception, even though processing for a machine check is run using the interrupt stack. IPL 0 is essentially used only to indicate that no interrupts are pending or being handled.

3.1.4.3 System Control Block

The SCB is the data structure used on a VAX to dispatch control to the appropriate service routine to handle exceptions and interrupts. The SCB is one page in length and is pointed to by the SCB base register (which contains the physical base address of the SCB). There is a vector in the SCB for each entity directly connected to the system backplane. For some processors, the SCB contains additional pages for I/O device interrupt routines. In addition to the address of the routine to handle the interrupt or exception of interest (loaded by software), information in the SCB will indicate whether processing for the event is to be done using the kernel or interrupt stack or whether writable control store is used.\[^5\]

3.1.4.4 Disabling Exceptions

Most exceptions cannot be disabled. They are handled immediately, regardless of the executing priority of the process that generated the exception. However, some trap conditions such as trace and overflow detection may be disabled by any software through the use of two instructions, Bit Clear PSW (BICPSW) and Bit Set PSW (BISPSW).

\[^5\] Writable control store is not used in the evaluated configuration.
3.1.4.5 Interrupt and Exception Processing

Interrupts are serviced between instructions or at predefined points in long, iterative instructions, such as string instructions. Interrupt or exception handling is initiated when the appropriate vector is read from the SCB (via hardware). The current PSL is saved and a new PSL is created and loaded. In the case of an interrupt, the IPL is raised to the level of the request. If the vector indicates that processing should be done using the interrupt stack (the two least significant bits are set to 01 binary), the IS bit is set in the PSL and the IPL is raised to 31, temporarily blocking all interrupts. If the low bits do not contain 01, then the setting of the IS bit is unchanged from the old PSL. The old stack pointer is saved and the new one is loaded, unless the process is already running on the interrupt stack. The old PSL and program counter are then pushed onto the stack. If the interrupt is a software interrupt, then the bit indicating a software interrupt is pending is cleared. The program counter is then loaded with the address of the service routine contained in the SCB vector.

For change mode instruction processing, the current mode (CUR_MOD) is loaded with the new mode; the previous mode (PRV_MOD) is loaded with the (now) old value of CUR_MOD; the IS bit and the IPL are kept from the old PSL; and all other fields in the PSL are cleared. For instruction emulation, CUR_MOD, PRV_MOD, the IS bit, IPL, and the condition codes are retained from the old PSL and all other fields are cleared. For any other type of exception, PRV_MOD is loaded with the old value of CUR_MOD. The IS bit and IPL are loaded according to the SCB vector (according to the contents of the low bits as described above). All the other fields in the PSL are zeroed out. As is the case during interrupt processing, unless the process is already executing on the interrupt stack, the old stack pointer is saved and the new one is loaded. The old PSL and the program counter are then pushed on the stack along with any exception parameters.

Exception processing is similar to that of interrupts initially. Before the appropriate vector is read from the SCB, the current PSL, the Program Counter (PC) that points to the instruction that caused the exception, and any exception parameters are pushed onto a stack. In most cases, initial exception processing is performed in kernel mode, using the kernel mode stack.\footnote{An example of an exception that doesn't use the kernel mode stack is one that occurs during execution of the change mode instruction handlers.} Hereafter, OpenVMS/SEVMS performs exception processing. An exception service routine pushes an exception type onto the stack, adding to the information previously pushed onto the stack by hardware. A general purpose exception routine is then entered to push additional information, such as particular register values, onto the stack. If it is determined that the exception should be processed by a handler, then the information on the stack is copied onto the stack of the exception-causing hardware access mode. Otherwise, a fatal exception is generated. In the former case, after the stack information has been copied, an REI instruction is executed causing the system to enter another general purpose routine in the mode in which the exception occurred. This routine builds a condition handler argument list that contains pointers to the information previously copied onto the (now current) exception-causing hardware access mode stack. This same routine also checks to see whether the exception was caused during instruction emulation. If so, execution is transferred to a particular system routine to handle that case (the PC is changed to point to the instruction being emulated and the data pushed onto the stack during instruction emulation is removed). Otherwise, execution is transferred to a routine that determines where the appropriate access mode condition handler (software) is located.

Instruction emulation is accomplished using one of two exceptions; the reserved/privileged instruction exception or a special subset instruction emulation exception. The first method is supported on all processors. On a processor that uses instruction emulation, the SCB is altered by OpenVMS/SEVMS to point to an emulation routine that executes prior to the normal reserved instruction exception service routine.
emulation routine checks the opcode of the instruction causing the exception. If it is one that is determined to be emulated, then it is processed by the emulation routine. Otherwise, the exception is passed on to the normal service routine.

For some processors, such as the MicroVAX II and the VAX 6000 series, a special instruction emulation exception is supported, mainly for character and decimal string instructions. When the microcode detects that an instruction to be executed is not in the defined set, exception parameters and appropriate operands are pushed onto the current hardware access mode stack. Execution is then transferred to a general instruction emulation routine without changing hardware access mode.

Interrupt service routines and exception handlers terminate by executing an REI instruction. The REI instruction processing is as follows. The saved PSL and program counter are popped from the stack. The PSL is checked for consistency (to make sure the mode transition is valid). The current stack pointer is saved and a new stack pointer is loaded according to the loaded PSL (CUR.MOD and IS fields). The service routine restores any registers or parameters that were saved. The level of the most privileged AST is checked against the current hardware access mode to see whether a pending AST can be delivered. Execution then resumes at the instruction immediately following that which was being executed at the time the exception or interrupt was detected (or the instruction that was interrupted is restarted or continued, if it can be).

### 3.1.5 Memory Management

This section discusses the hardware memory management on the VAX. The virtual address space structure is presented, along with fault processing and the address translation mechanism.

Memory management, of course, is handled by both software and hardware. On a VAX, the hardware provides page mapping and page protection. The page protection is provided by associated codes that determine the access based on the process’s access mode. The hardware access modes were discussed previously on page 13, “Architecture Overview”. Page faults are handled as exceptions (see page 23, “Interrupts and Exceptions”). In addition, memory management employs several of the internal processor registers discussed on page 14, “Internal Processor Registers”. The software functionality provided by OpenVMS/SEVMS involves creation of processes and paging. Both a virtual page and a physical page are 512 bytes long.

#### 3.1.5.1 Physical Address Space

As mentioned on page 13, “Architecture Overview”, the physical address space is divided into that reserved for I/O space and that which is used to contain executable code and program data such as OpenVMS/SEVMS and application programs, called memory space. All of physical memory is addressable as Random Access Memory (RAM); however, not all of physical memory is actually backed by a memory board. I/O space is backed by several entities, such as a device or a bus adapter.

Some processors support a 32-bit physical address (some of the 6000 series and the 9000 series). The amount of memory that can be configured, though, is limited by the number of available slots for boards, not the bit range. The limit on physical memory is currently 3.5 Gbytes. The entire physical 4 Gbyte address range is not available. The virtual address of a process is not increased with the 32-bit physical address functionality.
3.1.5.2 Virtual Address Space

A process’ virtual address space maximum size is 4 Gbytes. The virtual address space of a process is divided into what is called system space and process space. Both system space and process space are further divided into sections called S0 and S1 (system) and P0 and P1 (process). System space is the same for all processes; process space is unique to each process. For more information on how software uses the address regions, see page 87, “Memory Management”. Figure 3.2 depicts the virtual address space structure of a process.

A virtual address is 32 bits long. Most VAX processors use a 30-bit physical address. As noted in Figure 3.3, bits 9:31 contain the virtual page number. Bits 30 and 31 are used to indicate the address space region. Bits 9:29 are used to obtain a PTE address. Page tables are discussed in the next section. Bits 0:8 contain the byte offset within a page.
3.1.5.3 Page Tables

A PTE contains four pieces of information about a memory page: the protection code, the page frame number, the valid bit, and the modification bit. The format of a PTE is depicted in Figure 3.4.

There are two page tables kept for every process, one for each of the process address space regions. There is one page table for the system space region. There is a length register for each page table that contains the number of pages currently used in the respective region. The system page table is kept in contiguous pages of physical memory. The base register for the system page table contains the physical starting address of the system page table. The process page tables are allocated in contiguous pages of virtual system space. The base registers for the process page tables contain virtual addresses. Figures 3.5 and 3.6 depict the address translation algorithms for system and process addresses, respectively (address translation is described in more detail on page 30, “Virtual Address Translation”).

There is a translation buffer that is a cache for recently used page table entries. The address translation for process space addresses involves two memory references, assuming translation buffer misses. The first memory reference occurs during translation of a process virtual address into the system virtual address of the process PTE; the second occurs during the translation of the system virtual address into the physical address of the process PTE (lookup of the PTE that maps the page containing the process PTE). In practice, two memory references often are not needed. The translation buffer will most likely contain at least the process virtual address of the appropriate page table, thereby saving at least one memory reference.

The protection code describes the type of access allowed for each hardware access mode. Table 3.3 describes the types of accesses that may be defined. Hardware enforces the access specified in the protection codes on each reference to a location inside a page.

The valid bit, when set, indicates that both the setting of the modify bit and the page frame number are valid and that interpretation of the information in the PTE is hardware based. When the valid bit and the modify bit are set, the page has been modified. The modify bit is set by the CPU upon a successful write to the page. System software may also set the bit during modify fault processing. The modify bit is cleared
Table 3.3. Page Table Entry Protection Codes (VAX)

<table>
<thead>
<tr>
<th>Decimal No.</th>
<th>Operation</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Kernel</td>
</tr>
<tr>
<td>0</td>
<td>No Access</td>
<td>none</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
<td>n/a</td>
</tr>
<tr>
<td>2</td>
<td>Kernel Write</td>
<td>write</td>
</tr>
<tr>
<td>3</td>
<td>Kernel Read</td>
<td>read</td>
</tr>
<tr>
<td>4</td>
<td>User Write</td>
<td>write</td>
</tr>
<tr>
<td>5</td>
<td>Executive Write</td>
<td>write</td>
</tr>
<tr>
<td>6</td>
<td>Executive Read, Kernel Write</td>
<td>write</td>
</tr>
<tr>
<td>7</td>
<td>Executive Read</td>
<td>read</td>
</tr>
<tr>
<td>8</td>
<td>Supervisor Write</td>
<td>write</td>
</tr>
<tr>
<td>9</td>
<td>Supervisor Read, Executive Write</td>
<td>write</td>
</tr>
<tr>
<td>10</td>
<td>Supervisor Read, Kernel Write</td>
<td>write</td>
</tr>
<tr>
<td>11</td>
<td>Supervisor Read</td>
<td>read</td>
</tr>
<tr>
<td>12</td>
<td>User Read, Supervisor Write</td>
<td>write</td>
</tr>
<tr>
<td>13</td>
<td>User Read, Executive Write</td>
<td>write</td>
</tr>
<tr>
<td>14</td>
<td>User Read, Kernel Write</td>
<td>write</td>
</tr>
<tr>
<td>15</td>
<td>User Read</td>
<td>read</td>
</tr>
</tbody>
</table>

only by software.

When the valid bit is reset, it indicates that software intervention is required to interpret the meaning of the information in the PTE. This implies that OpenVMS/SEVMS is using a combination of the valid bit and the software bits (bits 21:22 and the modify bit) in the PTE to implement other memory management functions on a page. Examples of those functions are: zero a page, copy on reference, page sharing, and transitions between active and swapped-out states. These functions are implemented whenever a page fault occurs, which results whenever a reference is made to a PTE when the valid bit is not set. Bits 21:22 are also used as the page frame number extension bits when 32-bit physical addressing is in effect.

The page frame number contains the physical memory base address of the virtual page. The page frame number is used by the hardware in virtual to physical address translation. Address translation is described in the following section.

The owner bits are not used by hardware. The access mode of the owner of the page is kept in this field. The field is used to indicate in what mode the page protection can be altered or deleted. Bit 25 is reserved and is always zero.

PTEs are modified through move or interlocked instructions. A move instruction is used to modify several PTE fields at one time, in order to avoid collisions. Interlocked or atomic operation instructions are used when the PTE contents are consistent after execution of one instruction. The hardware ensures that the move instruction is uninterruptible to prevent inconsistencies between multiprocessors.

3.1.5.4 Faults

There are two types of faults associated with memory management; translation-not-valid and access-control-violation. These faults are handled as exceptions: that is, control is passed to an appropriate exception
Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha

CHAPTER 3. HARDWARE ARCHITECTURE

handler. A translation-not-valid fault occurs when a reference is made to a PTE marked invalid. As a result of this type of fault, a fault parameter word is created that contains the virtual address of the instruction which caused the fault. The fault parameter word is pushed onto the kernel stack as part of the page fault processing.

An access-control-violation fault occurs when the requested access to a page is denied by checking the protection field in the PTE or when a reference is beyond the end of a page table (length violation). In this case, the fault parameter word contains the intended access for the page and whether or not the fault results from bounds checking.

Vector processors employ one of two methods in handling memory management exceptions (primarily page faults): synchronous and asynchronous. The synchronous method implies that no new scalar or vector instructions can execute until the memory access vector instruction is guaranteed to execute without incurring any memory management exceptions. When a memory management exception occurs using the synchronous method, the vector instruction is stopped and the exception is handled by VAX memory management instruction processing. If the exception is repaired, the vector instruction is eventually reexecuted.

With the asynchronous method, the scalar processor continues to process instructions when a vector processor memory management exception occurs. In this case, all pending vector instructions are allowed to complete as long as the associated source data is valid. The scalar processor is notified of a vector memory management exception when the next vector instruction is passed to the vector processor. The VAX memory fault processing is then invoked. If the exception is repaired, the vector processor then retries the faulting memory reference instruction in the state in which the processor was executing when the fault occurred (using the vector state address register). Processing then continues with the vector instruction sent over from the scalar processor.

3.1.5.5 Virtual Address Translation

Virtual address translation is controlled by the setting of the Memory Map Enable bit contained in the internal Memory Management Enable register. The Memory Map Enable bit is always set during normal operation of OpenVMS/SEVMS but it is clear while bootstrap procedures are executing. When the Memory Map Enable bit is set, translation of virtual to physical addresses is performed and page protection is enforced.

As mentioned previously, there are three page tables used to perform virtual address translation: one for system space and two for process space. Figure 3.5 depicts the address translation algorithm for a system address. First, bits 9:29 of the virtual address are extracted and checked to see if the page number is within the bounds of the system region. This is done using the system base register and the system length register, that contain the physical address of the system page table and the number of PTEs, respectively. Once the bounds check is performed and the page number validated, the contents of the system page table base register is added to the virtual page number to get the physical address of the PTE of interest (this is done in kernel mode). The contents of the PTE are examined and checked against the requested hardware access mode of the page. If the access mode check passes, bits 0:8 of the original virtual address are appended to the page frame number in the PTE to get the physical address of the data.

For process address translation, the procedure depicted in Figure 3.5 describes the “Fetch” and “Check Access” steps in Figure 3.6. The algorithm is essentially the same as that described above with the extra steps of retrieving the P0 page table address and performing bounds checking using the P0 base and length registers.
Figure 3.5. System Virtual to Physical Address Translation (VAX)
In the case of 32-bit physical addressing, the page frame number is extended from 21 to 23 bits (bits 0:22 of the PTE). The physical memory capacity is increased to 3.5 Gbytes, and the interpretation and usage of the reserved bits in the PTE is changed. A system flag, located in low memory and set in a conditionally loaded executive image (see page 73, “VMS Executive”), is used to indicate whether 30 or 32-bit addressing is in use for the processor.

For P1 space translation, the P1 base register and P1 length register are used. The P0 and P1 regions are allocated in opposite directions; that is, the P0 region is allocated toward numerically higher addresses, whereas the P1 region is allocated toward numerically lower addresses. The P1 base and length registers describe the portions of P1 space not used to keep the address translation algorithm consistent across regions. The address in the P1 base register does not have to be in system space, but the address of any fetched P1 space PTE must be within the bounds of system space.

### 3.1.6 I/O Management

There are two different methods used to perform I/O. Most of the processors use I/O space. Others use the mailbox I/O mechanism.

#### 3.1.6.1 I/O Space

Each I/O device controller has a corresponding set of control and status and data registers located at assigned addresses in physical address space. The registers are mapped into virtual S0 space and are addressed by the device drivers using the normal VAX instruction set (all of physical memory is memory mapped). These registers are used by the driver to issue commands to the controller.

Physically, I/O space is not memory, but is a set of locations on an I/O bus adapter or backplane board. Some I/O adapters perform an additional address translation step to send a command to the device to be used. Other adapters, those used in Direct Memory Access (DMA) operations, contain a set of registers used to hold the physical memory address of a data location. Some I/O busses can be used to serve both memory and devices. In that case, the address space provided by the bus is divided into memory address space and I/O space. Devices on such I/O busses can use PTEs to translate addresses. PTEs used by I/O devices have a specific format and are not used by the hardware (I/O PTEs are marked invalid).

I/O interrupt handling is controlled by software. Device drivers control interrupt enabling and disabling for the controllers for which the driver is responsible. An interrupt occurs when the controller requests it, while interrupts are enabled. The processor accepts the interrupt request and executes the device driver’s interrupt service routine when the priority level lowers to that of the device. For details on software processing of an I/O interrupt, see page 123, “Device Drivers”.

#### 3.1.6.2 Mailbox I/O

The VAX 7000 and 10000 models perform I/O through the use of a mailbox structure. The mailbox is used to access the control and data registers of the device to which I/O is issued. The mailbox contains the bus address of the control status register of the device, information describing the operation to be performed, and flags. Mailboxes are allocated from non-paged memory (in S0 space) by the drivers.
Figure 3.6. P0 Virtual to Physical Address Translation (VAX)
With mailbox I/O, the I/O bus cannot be directly accessed via physical memory, as in the use of I/O space. The I/O bus is remotely configured. During an I/O operation, the address of the mailbox is passed to a hardware component called an I/O Port Module (IOP) that forwards the address to a remote bus adapter. The internal mailbox pointer register is written by the device driver (via the MTPR instruction) and a "work request" is posted to the IOP. The IOP services the mailbox when it has completed its current task. The IOP fetches the mailbox structure and builds a command packet that is passed down a "hose" to the bus adapter. The command packet contains the actual I/O command and device address. The adapter is configured as a separate device on the remote I/O bus. The command to be performed is passed to the device over the I/O bus. Once the operation is complete, the IOP merges the status information returned from the bus adapter into the main memory mailbox.

### 3.2 Alpha Architecture

This section describes basic concepts and capabilities of the Alpha architecture. An overview of the processor is presented followed by discussions of the instruction set, process management, interrupts and exceptions, memory management, and I/O management.

#### 3.2.1 Architecture Overview

The Alpha architecture, which is a 64-bit load/store Reduced Instruction Set Computing (RISC) architecture, is designed to support a highly flexible, multiprogramming environment. Multiprogramming is supported through context switching, process isolation, virtual addressing, memory management, and hardware access modes that support hierarchical execution domains.

All Alpha registers are 64-bits in length and all operations are performed between 64-bit registers. All instructions are 32-bits in length, memory operations are either loads or stores, and all data manipulation is done between registers. A byte addressable 64-bit virtual address space is supported. There are 32 interrupt levels. The Alpha architecture facilitates pipelining multiple instances of the same operations because there are no special registers and no condition codes. The Alpha architecture is designed to accommodate several operating systems, two of these being OpenVMS and SEVMS. To accomplish this, Alpha is comprised of two components, the common architecture component and an operating system specific component. The former is common to all operating systems running on Alpha and is implemented in hardware. The later, implemented in software, is called PALcode. PALcode instructions are used to match specific operating system requirements to the underlying execution domains.

For OpenVMS and SEVMS running on Alpha, PALcode instructions are used to switch the hardware context of a process structure, implement uninterruptable queue operations, and to provide interrupt and exception mechanisms that are independent of the underlying hardware implementation.

Alpha uses a "mailbox I/O" mechanism for I/O. A mailbox contains the bus address of the control status register of the device as well as information describing the operation to be performed and flags. This mechanism, which is also used in VAX 7000 and 10000 systems, is described on page 32, "Mailbox I/O".
3.2.2 Registers and Instruction Set

This section describes Alpha registers and the Alpha instruction set.

3.2.2.1 Registers

There are 32 integer registers (R0 through R31), each 64-bits wide. R31 reads as zero, and writes to R31 are ignored. In addition, there are 32 floating-point registers (F0 through F31), each 64-bits wide. F31 reads as zero, and writes to F31 are ignored.

All integer data manipulation is between integer registers, with up to two variable register source operands, and one register destination operand. Similarly, for floating-point manipulation, all floating-point data manipulation is between floating-point registers, with up to two register source operands and one register destination operand.

Within Alpha, all memory reference instructions are of the load/store type that move data between registers and memory. There are no branch condition codes. Branch instructions test an integer or floating-point register value, which might be the result of a previous compare. Integer and logical instructions operate on quadwords. Floating-point instructions operate on G-floating, F-floating, IEEE double, and IEEE single operands.

The program counter (PC) is a special register that addresses the instruction stream. As each instruction is decoded, the PC is advanced to the next sequential instruction. The PC is not accessible as a register. Integer register R30 is the stack pointer (SP). The SP contains the address of the top of the stack in the current mode.

Additional Alpha registers are identified and discussed later in this section (see page 38, "Additional Alpha Registers").

3.2.3 Instruction Set and PALcode Environment

This section describes the instruction set for Alpha. In addition, the PALcode environment is discussed. PALcode, which is found within the Privileged Architecture Library (PAL), is used to specify extended processor functions.

3.2.3.1 Instruction Set Characteristics

All Alpha instructions are 32-bits in length. There are four Alpha instruction format classes: memory format (two register fields); operate format (three register fields); branch format (one register field); and Privileged Architecture Library code (PALcode) format (no register fields). These are shown in Figure 3.7.

Load/store instructions belong to the memory format class. These instructions move longwords or quadwords between a designated register, Ra, and memory, using a designated register, Rb, and a signed 16-bit displacement as the memory address. There are facilities for doing byte manipulation in registers, thus eliminating the need for 8-bit and 16-bit load/store instructions. A 32-bit integer datum is placed into a register in a canonical form that makes 33 copies of the high bit of the datum.
The integer operate instructions, which belong to the operate format class, manipulate full 64-bit values, and include arithmetic, compare, logical, and shift instructions. Integer operations use registers Ra and Rb as the source operand, and write the result in register Rc. There are just three 32-bit integer operations: add; subtract; and multiply. They differ from their 64-bit counterparts only in overflow detection and in producing 32-bit canonical results. There is no integer divide instruction. A minimal number of VAX compatibility instructions are included.

Conditional branch instructions, which belong to the branch format class, can test the contents of a register for positive, negative, or zero. They can also test for even or odd. Unconditional branch instructions can write a return address into a register. There is also a calculated jump instruction that branches to an arbitrary 64-bit address.

### 3.2.3.3 The PALcode Environment

In Alpha, some of the functions normally required by an operating system are implemented in native mode directly, and others, which are impractical to implement directly in a RISC system, are implemented using PALcode. Some of the functions that involve PALcode are low-level hardware support functions (e.g., Translation Buffer miss fill routines, interrupt acknowledge), support for privileged operations, and support for atomic operations that require long instruction sequences. In CISC systems, these additional functions are typically implemented in microcode. The Alpha system does not include any microcode execution, so these functions are implemented using the PALcode environment.

The PALcode environment differs from the normal environment (i.e., native mode) in the following ways:

- Complete control of the machine state
- Interrupts are disabled
- Instruction-stream memory mapping is disabled
- Access to special hardware registers and functions (these may differ from processor to processor)

Complete control of the machine state allows all functions of the machine to be controlled. Disabling inter-
3.2. ALPHANARCHITECTURE

Interrupts allows the system to provide multi-instruction sequences as atomic operations. Preventing instruction-stream memory management traps allows PALcode to implement memory management functions such as translation buffer fill. Finally, providing access to all special hardware registers and to special instructions allows access to low-level hardware.

PALcode "routines" (i.e., the sequences of instructions that are invoked in the PALcode environment when a CALLPAL instruction is executed, or an exception condition is generated) typically use native mode Alpha instructions. Some additional functions are needed which are provided by five Alpha instructions that can be executed only in the PALcode environment; if execution is attempted in native mode, an "invalid instruction" exception is generated. These special instructions permit access to the additional registers not visible to native mode programs and provide special access to memory.

PALcode is brought into system memory during system initialization, and executes from that memory. The operating system must protect that memory so that it is not modified either by the operating system or by any programs outside the operating system.

Every implementation using an Alpha processor must provide some minimum PALcode functions (halting the processor, insuring instruction and data stream coherence, and draining interrupts). Beyond that, the implementor is free to develop whatever support functions are desired. In the case of SEVMS and OpenVMS, the PALcode is called "OpenVMS Alpha software", and supports most of the VAX architectural functions.

The PALcode that is executed is loaded from ROM that is shipped with the CPU board. PALcode ROM is only read at boot time. Once loaded into memory, the operating system prevents any process from modifying the PALcode. Also, the operating system cannot access the ROM.

The PALcode environment can be entered only by execution of the CALL_PAL instruction, or by interrupts and exceptions. In either case, protection is obtained because the PALcode is in control, and was entered through a controlled entry. A native-mode program can not execute the special instructions that are available only to the PALcode environment because the Alpha processor permits execution of these instructions only in the PALcode environment, and, as indicated above, it can be entered only in a controlled manner.

For OpenVMS and SEVMS running on Alpha, there are twenty-one unprivileged PALcode instructions and eight privileged PALcode instructions. Examples of the unprivileged PALcode instructions are AMOVRM (atomic move register/memory), BPT (breakpoint), CHMU (change mode to user), RD_PS (read processor status), and REI (return from exception or interrupt). The eight privileged PALcode instructions are CFLUSH (cache flush), DRAIN (drain aborts), HALT (halt processor), LDQP (load quadword physical), MFPR (move from processor register), MTPR (move to processor register), STQP (store quadword physical), and SWPCTX (swap privileged context).

3.2.3.3 Parameter Validation Instructions

There are two unprivileged PALcode instructions, PROBER and PROBEW, that are used by OpenVMS/SEVMS software to verify that a less privileged caller has read or write access to the parameters that are passed to a service routine. These routines are used to check the read and write accessibility of the first and last byte specified by the base address and signed offset (the bytes in between are not checked). System software must check all pages between the two bytes if they are to be accessed.
3.2.3.4 PALcode Processor Modes

In the OpenVMS (and SEVMS) PALcode environment there are four access modes. They are, ranging from most to least privileged: kernel, executive, supervisor, and user. Privileged instructions can only be executed in kernel mode. The memory is protected using protection codes for each memory page. For each access mode, a page may be readable, readable and writable, or inaccessible. If a page is accessible in a particular access mode, it is also accessible in the modes that are more privileged than that particular mode. For example, if a page is readable in supervisor mode, it is also readable in executive and kernel modes.

The change mode instructions allow a user to explicitly change from one hardware access mode to another, thereby providing access to protected code and data. Change mode instructions can only be issued to progress to a mode equal to or more privileged than the current mode. They can be issued from any mode. However, if an illegal operation is attempted, such as executing a CALL.PAL CHMS (change mode to supervisor) instruction from executive mode, then no change in mode results and an access-control-violation fault occurs. This check is done before any other change mode processing takes place.

Change mode instructions cause exceptions to be generated that are handled by service routines configured to execute for the specified mode change. This is how system services are implemented in OpenVMS/SEVMS (see page 79, “System Services”).

3.2.3.5 Additional Alpha Registers

The Processor Status Register

The processor status (PS) is a special 64 bit register that contains the current status of the processor. The PS is shown in Figure 3.8 and a description of the PS fields is shown in Table 3.4. The PS can be explicitly read with the CALL.PAL RD_PS instruction and the PS(SW) field can be explicitly written with the CALL.PAL WR.PS_SW instruction. The PS(SW) field is reserved for software use and can be read and written at any time by the software, regardless of the current mode; the value of these bits is ignored by the hardware. The value of this field is set to zero at the initiation of either an exception or an interrupt. The current mode (CM) field identifies the access mode of the currently executing process (0, kernel; 1, executive; 2, supervisor; 3, user). The IPL field indicates the current processor priority in the range of 0 to 31. At bootstrap, the initial value of PS is set to 1F00 (hex); previous stack alignment is set to zero; IPL is set to 31; CM is set to kernel; and the SW, VMM, and IP fields are zero.

Internal Processor Registers

Alpha has internal processor registers (IPRs) that provide architectured mapping to internal hardware. These registers are available only through privileged PALcode routines (MFPR and MTPR) and allow OpenVMS or SEVMS to interrogate or modify system states. Several of the internal processor registers are loaded during a context switch when a process transitions from the ready state to the running state. Table 3.5 lists several of the internal processor registers.

The System Control Block Base (SCBB) register holds the Page Frame Number (PFN) of the System Control Block (SCB), which is used to dispatch exceptions and interrupts. The Privileged Context Block Base (PCBB) register contains the physical address of the privileged context block for the current process. The Page Table Base Register (PTBR) contains the page frame number of the first-level page table for the current process. The stack pointers for user, supervisor, and executive stacks are accessible as IPRs through the privileged CALL.PAL MTPR and MFPR instructions.
3.2. ALPHA ARCHITECTURE

Figure 3.8. Processor Status Register (Alpha)

<table>
<thead>
<tr>
<th>Bit Extent</th>
<th>Name</th>
<th>Mnemonic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 to 62</td>
<td>Reserved</td>
<td>MBZ</td>
<td>Reserved by Digital; must be zero.</td>
</tr>
<tr>
<td>61 to 56</td>
<td>Stack Alignment</td>
<td>SP.ALIGN</td>
<td>Used by the CALLPAL REI instruction to restore the previous stack byte alignment.</td>
</tr>
<tr>
<td>55 to 13</td>
<td>Reserved</td>
<td>MBZ</td>
<td>Reserved by Digital; must be zero.</td>
</tr>
<tr>
<td>12 to 8</td>
<td>Interrupt Priority Level</td>
<td>IPL</td>
<td>Contains the current processor priority (in the range of 0 to 31).</td>
</tr>
<tr>
<td>7</td>
<td>Virtual Machine Monitor</td>
<td>VMM</td>
<td>When set, the processor is executing in a virtual machine monitor. When clear, the processor is running in either real or virtual machine mode.</td>
</tr>
<tr>
<td>6 to 5</td>
<td>Reserved</td>
<td>MBZ</td>
<td>Reserved by Digital; must be zero.</td>
</tr>
<tr>
<td>4 to 3</td>
<td>Current Mode</td>
<td>CM</td>
<td>Contains the access mode of the currently executing process. The modes are 0 (Kernel); 1 (Executive); 2 (Supervisor); and 3 (User).</td>
</tr>
<tr>
<td>2</td>
<td>Interrupt Pending</td>
<td>IP</td>
<td>This bit is set when a software or hardware interrupt (but not an AST) is initiated, indicating that an interrupt is in progress.</td>
</tr>
<tr>
<td>1 to 0</td>
<td>Reserved for Software</td>
<td>SW</td>
<td>The bits are reserved for software use and can be read and written by the software at any time regardless of the current mode. The software field is set to zero at the initiation of either an exception or an interrupt.</td>
</tr>
</tbody>
</table>

Table 3.4. PS Description (Alpha)
Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha

CHAPTER 3. HARDWARE ARCHITECTURE

<table>
<thead>
<tr>
<th>Name</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address Space Number</td>
<td>ASN</td>
</tr>
<tr>
<td>AST Enable</td>
<td>ASTEN</td>
</tr>
<tr>
<td>AST Summary Register</td>
<td>ASTSR</td>
</tr>
<tr>
<td>Data Align Trap Fixup</td>
<td>DATFX</td>
</tr>
<tr>
<td>Floating-point Enable</td>
<td>FEN</td>
</tr>
<tr>
<td>Interrupt Priority Level</td>
<td>IPL</td>
</tr>
<tr>
<td>Machine Check Error Summary</td>
<td>MCES</td>
</tr>
<tr>
<td>Privileged Context Block Base</td>
<td>PCBB</td>
</tr>
<tr>
<td>Processor Base Register</td>
<td>PRBR</td>
</tr>
<tr>
<td>Page Table Base Register</td>
<td>PTBR</td>
</tr>
<tr>
<td>System Control Block Base</td>
<td>SCBB</td>
</tr>
<tr>
<td>Software Interrupt Request Register</td>
<td>SIRR</td>
</tr>
<tr>
<td>Software Interrupt Summary Register</td>
<td>SISR</td>
</tr>
<tr>
<td>TB Check</td>
<td>TBCHK</td>
</tr>
<tr>
<td>Kernel Stack Pointer</td>
<td>KSP</td>
</tr>
<tr>
<td>Executive Stack Pointer</td>
<td>ESP</td>
</tr>
<tr>
<td>Supervisor Stack Pointer</td>
<td>SSP</td>
</tr>
<tr>
<td>User Stack Pointer</td>
<td>USP</td>
</tr>
<tr>
<td>Virtual Page Table Base</td>
<td>VPTB</td>
</tr>
<tr>
<td>Who-Am-I</td>
<td>WHAMI</td>
</tr>
</tbody>
</table>

Table 3.5. Internal Processor Registers (Alpha)

3.2.3.6 Asynchronous System Trap (AST)

ASTs are a means of notifying a process of events that are not synchronized with its execution, but which must be dealt with in the context of the process with minimum delay. ASTs interrupt process execution and are controlled by the AST Enable (ASTEN) and AST Summary (ASTSR) internal processor registers. The ASTEN register contains an enable bit for each of the four processor access modes. When the bit corresponding to an access mode is set, an AST is pending for that mode. ASTs that are pending and enabled for a less privileged access mode are not allowed to interrupt execution in a more privileged access mode.

3.2.4 Process Management

This section discusses the definition of a hardware process context and how context switching is performed on an Alpha processor.

3.2.4.1 Process Context

A process is the entity that is scheduled for execution by the processor. A process represents a single thread of execution and consists of an address space, and hardware and software context.

For Alpha, the hardware context is defined by the following:

- The 32 integer registers
3.2. ALPHA ARCHITECTURE

- The 32 floating-point registers
- Processor Status (PS)
- Program Counter (PC)
- Four stack pointers
- Asynchronous System Trap Enable and summary registers (ASTEN, ASTSR)
- Process Page Table Base Register (PTBR)
- Address Space Number (ASN)
- Floating-point Enable Register (FEN)
- Process Cycle Counter (PCC)
- Process Unique value
- Data Alignment Trap (DAT)
- Performance Monitoring Enable register (PME).

For Alpha, the software context of a process is defined by the operating system software; in this case OpenVMS or SEVMS.

3.2.4.2 Context Switching

For Alpha, process context switching occurs as one process after another is scheduled for execution by OpenVMS or SEVMS system software. Context switching requires the hardware context of one process to be saved in memory, followed by the loading of the hardware context for another process into the hardware registers.

The privileged hardware context is swapped with the CALL.PAL Swap Privileged Context instruction (SWPCTX). This instruction returns ownership of the current Hardware Privileged Context Block (HWPCB) to the operating system and passes ownership of the new HWPCB to the processor. The privileged context includes the four stack pointers, the Page Table Base Register (PTBR), the Address Space Number (ASN), the AST enable and summary registers, the Floating-point Enable register (FEN), the Performance Monitoring Enable (PME) register, the Data Alignment Trap register (DAT), and the Process Cycle Counter (PCC). SWPCTX, an atomic operation, always saves the privileged context of the old process and loads the privileged context of a new process.

3.2.4.3 System Control Block

The SCB is the data structure used by Alpha to specify entry points for exception, interrupt, and machine check routines. The block is from 8K to 32K in length. The SCB consists of from 512 to 2948 entries, each 16 bytes in length. The first 8 bytes of an entry (the vector) specify the virtual address of the service routine associated with that entry. The second 8 bytes (the parameter) contains an arbitrary quadword value that is to be passed to the service routine. The SCB entries fall into one of the following groups:

- Faults
- Arithmetic traps
- Asynchronous system traps
- Data alignment trap
- Other synchronous traps
- Processor software interrupts
CHAPTER 3. HARDWARE ARCHITECTURE

- Processor hardware interrupts
- I/O device interrupts
- Machine checks.

The first 512 entries contain all architecturally defined and any statically allocated entries. Additional SCB entries, if any, are used for I/O device interrupt vectors that are assigned dynamically by system software.

3.2.5 Interrupts, Exceptions, and Machine Checks

Interrupts, exceptions, and machine checks are events that change the flow of processing on an Alpha processor. Interrupts are events that are systemwide in scope and are handled in “system context”. Exceptions are events that affect the executing process and are handled by OpenVMS/SEVMS, usually in the “process context”. Machine checks are events that result from serious hardware failure.

3.2.5.1 Exceptions

Exceptions are events that are primarily relevant to the currently executing process and normally invoke software in the context of the current process. There are three types of exceptions: faults, arithmetic traps, and synchronous traps. A fault is an exception condition that occurs during an instruction and leaves the registers and memory in a consistent state such that elimination of the fault condition and subsequent re-execution of the instruction will give correct results. An arithmetic trap is an exception condition that occurs at the completion of the operation that caused the exception. A synchronous trap is an exception condition that occurs at the completion of the operation that caused the exception.

Most exception service routines execute in kernel mode, on the kernel stack, and all exception service routines execute at the current processor IPL. Change mode exception routines for CHMU, CHMS, and CHME execute in the more privileged of the current mode or the target mode on the matching stack. Exception service routines are usually coded to avoid exceptions.

3.2.5.2 Interrupts

Each Alpha processor has 32 interrupt priority levels (IPLs) divided into 16 software levels (0-15) and 16 hardware levels (16-31). If the priority of an interrupt is greater than that of the currently executing process, the processor will raise the priority level to that of the requested interrupt and the interrupt service routine is entered at the IPL of the interrupting source, in kernel mode, and on the kernel stack. When a particular interrupt level is set, all interrupts of lower priority are blocked from service. Interrupts at a particular level can be disabled by setting the IPL to a level equal to or higher than the level to be disabled. Bits 12:8 of the Processor Status (PS) register contain the IPL field. These bits can only be set through the privileged PAL code MTPR instruction.

Interrupts can come from devices, controllers, other processors, or the processor itself. A device interrupt is hardware-generated, as is a processor interrupt. An example of a processor interrupt is the AST Level interrupt which is handled at IPL 2. Kernel mode software can alter the priority of the processor it runs on, but not that of other processors. Kernel mode software also cannot disable interrupts on other processors. Interrupts are processed at the completion of the currently executing instruction or at a pre-defined point.
3.2. ALPHA ARCHITECTURE

during execution where the process state can be contained in the PS register, other registers, and the program counter.

As noted above, interrupt levels 0 through 15 are reserved for system software, and levels 16 through 31 for hardware. I/O device interrupts are handled at IPL 20-23; interval clock and interprocessor interrupts at IPL 22; performance monitor interrupt at IPL 29; and powerfail interrupt at IPL 30. IPL 0 is used to indicate that no interrupts are pending or being handled.

Interrupts are initiated in kernel mode and push the interrupt stack frame of eight quadwords onto the kernel stack. The PC saved in the interrupt stack frame is the virtual address of the first instruction not executed after the interrupt condition was recognized. A CALL.PAL REI instruction to the saved PC/PS will continue execution at the point of interrupt. Each interrupt source has a separate vector location (offset) within the System Control Block (SCB). With the exception of I/O device interrupts, each interrupt type identified in the previous paragraph has a unique vector. I/O device interrupts occupy a range of vectors that can be both statically and dynamically assigned. Upon entry to the interrupt service routine, R2 contains the SCB vector quadword and R3 contains the SCB parameter quadword. No memory mapping information is changed when an interrupt occurs.

3.2.5.3 Machine Checks

Hardware failure results in a machine check event. Registers and memory are potentially in an indeterminate state and instruction execution cannot necessarily be correctly restarted, completed, simulated, or undone. There are four types of machine checks: system machine check (IPL 31); processor machine check (IPL 31); system correctable machine check (IPL 20); and processor correctable machine check (IPL 31). Machine checks are initiated in kernel mode on the kernel stack, and cannot be disabled.

3.2.6 Memory Management

This section discusses Alpha memory management. The virtual address space structure is presented, along with fault processing and the address translation mechanism.

Alpha’s virtual memory space is made up of “System Space”, P1 space, and P0 space. “System Space” contains the executive, system-wide data structures, and any images installed permanently resident by the system administrator. P1 space contains the process stacks and permanent process control information maintained by the executive. It also contains address space used on the process’s behalf by inner access mode components such as Record Management Services, the file system, and a command language interpreter. P0 space maps whatever images the user activates.

3.2.6.1 Physical Address Space

For Alpha, physical addresses are at most 48 bits. A process may choose to implement a smaller physical address by not implementing some number of high order bits. All of physical memory is addressable as RAM. The DEC 7000 and 10000 systems support up to 3.5 Gbytes of RAM, and the AlphaServer 2100 supports up to 2 Gbytes of RAM.
3.2.6.2 Virtual Address Space

For OpenVMS and SEVMS Version 6.1 running on Alpha, the VAX architecture’s 32-bit address space is emulated. Figure 3.9 depicts Alpha’s virtual address space structure as it relates to OpenVMS and SEVMS. As can be seen, there is a 2 Gbyte process private address region (VAX P0 and P1) and a 2 Gbyte shared address region (VAX S0 and S1) for each process. For Alpha, the symbols P0 and P1 are used as they are for VAX. However, the phrase “System Space” is used in Alpha instead of the VAX symbols S0 and S1.

<table>
<thead>
<tr>
<th></th>
<th>P0 Space</th>
<th>P1 Space</th>
<th>Unused</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00000000 00000000</td>
<td>00000000 3FFFFFFF</td>
<td>00000000 00000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00000000 40000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>00000000 7FFFFFFF</td>
<td>00000000 80000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00000001 FFFFFFFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00000002 00000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00000003 FFFFFFFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00000004 00000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FFFFFFFF 7FFFFFFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FFFFFFFF 80000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FFFFFFFF FFFFFFFF</td>
</tr>
</tbody>
</table>

Figure 3.9. Virtual Address Regions (Alpha)

A virtual address is a 64-bit unsigned integer specifying a byte location within the virtual address space. Implementations subset the virtual address space supported to one of four sizes (43, 47, 51, or 55 bits). The minimal virtual address size supported is 43 bits. The virtual address space is broken into pages, which are the units of relocation, sharing, and protection. The page size ranges from 8K bytes to 64K bytes. The virtual address format is shown in Figure 3.10. The byte_within_page field can be either 13, 14, 15, or 16 bits depending on a particular page size implementation. Thus, the allowable page sizes are 8K bytes, 16K bytes, 32K bytes, and 64K bytes. The level-number fields are a function of page size.

3.2.6.3 Page Tables

The processor uses a quadword Page Table Entry (PTE) to translate virtual addresses to physical addresses. A PTE, which is always 64 bits in length, contains hardware and software control information and the
physical Page Frame Number. The Page Frame Number (PFN) field in the PTE is always 32 bits wide (bits 63:32). A PTE is shown in Figure refalpha-pte and the format of a PTE is shown in Table 3.6.

![Figure 3.10. Virtual Address Format (Alpha)](image)

![Figure 3.11. Page Table Entry (Alpha)](image)

### 3.2.6.4 Virtual Address Translation

The translation buffer (TB) is a CPU component that caches the result of recent successful virtual address translations of valid pages. Each TB entry caches one translation (a virtual page number and the corresponding PFN, address space match, and protection bits).

Figure 3.12 shows how address translation is performed. The basic steps are as follows.

- The Page Table Base Register (PTBR) points to the Level 1 Page Table (L1PT)
- The contents of the level 1 field (L1) in the virtual address indexes the L1PT to select a Level 1 Page Table Entry (L1PTE) which contains the PFN of a Level 2 Page Table (L2PT)
### Table 3.6. Page Table Entries (Alpha)

<table>
<thead>
<tr>
<th>Bit Extent</th>
<th>Name</th>
<th>Mnemonic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 to 32</td>
<td>Page Frame Number</td>
<td>PFN</td>
<td>The PFN always points to a boundary page.</td>
</tr>
<tr>
<td>31 to 16</td>
<td>Reserved for software.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>User Write Enable</td>
<td>UWE</td>
<td>Enables writes from user mode.</td>
</tr>
<tr>
<td>14</td>
<td>Supervisor Write Enable</td>
<td>SWE</td>
<td>Enables writes from supervisor mode.</td>
</tr>
<tr>
<td>13</td>
<td>Executive Write Enable</td>
<td>EWE</td>
<td>Enables writes from executive mode.</td>
</tr>
<tr>
<td>12</td>
<td>Kernel Write Enable</td>
<td>KWE</td>
<td>Enables writes from kernel mode.</td>
</tr>
<tr>
<td>11</td>
<td>User Read Enable</td>
<td>URE</td>
<td>Enables reads from user mode.</td>
</tr>
<tr>
<td>10</td>
<td>Supervisor Read Enable</td>
<td>SRE</td>
<td>Enables reads from supervisor mode.</td>
</tr>
<tr>
<td>9</td>
<td>Executive Read Enable</td>
<td>ERE</td>
<td>Enables reads from executive mode.</td>
</tr>
<tr>
<td>8</td>
<td>Kernel Read Enable</td>
<td>KRE</td>
<td>Enables reads from kernel mode.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved by Digital</td>
<td></td>
<td>Should not be used by software.</td>
</tr>
<tr>
<td>6 to 5</td>
<td>Granularity Hint</td>
<td>GH</td>
<td>Software may set these bits to a non-zero value to supply a hint to translation buffer implementations that a block of pages can be treated as a single larger page.</td>
</tr>
<tr>
<td>4</td>
<td>Address Space Match</td>
<td>ASM</td>
<td>When set, this PTE matches all Address Space Numbers.</td>
</tr>
<tr>
<td>3</td>
<td>Fault On Execute</td>
<td>FOE</td>
<td>When set, a fault on execute exception occurs on an attempt to execute an instruction in the page.</td>
</tr>
<tr>
<td>2</td>
<td>Fault on Write</td>
<td>FOW</td>
<td>When set, a fault on write exception occurs on an attempt to write any location on the page.</td>
</tr>
<tr>
<td>1</td>
<td>Fault on Read</td>
<td>FOR</td>
<td>When set, a fault on read exception occurs on an attempt to read any location on the page.</td>
</tr>
<tr>
<td>0</td>
<td>Valid</td>
<td>V</td>
<td>Indicates the validity of the PFN field. When V is set, the PFN field is valid for use by hardware.</td>
</tr>
</tbody>
</table>
• The contents of the level 2 field (L2) in the virtual address indexes the L2PT which contains the PFN of a Level 3 Page Table (L3PT)
• The contents of the level 3 field in the virtual address indexes the L3PT to select a L3PTE which contains the PFN of the code or data at that virtual address
• The contents of the byte within page field are concatenated with the PFN to form the target physical address.

![Virtual Address Translation Diagram](image)

**Figure 3.12. Virtual Address Translation (Alpha)**

### 3.3 VAX or Alpha System Node

A node (VAX or Alpha) consists of one or more processors, memory, a system bus, I/O busses, associated hardware (controllers and adapters), and peripheral devices. In the event that multiple processors are present in a node, they share memory and the local I/O devices. This section provides an overview of the hardware that can be associated with a VAX or Alpha node. A complete list of hardware included in this evaluation is contained in Appendix A of this report.

#### 3.3.1 Busses

There are two types of busses found in a node: the system bus and an I/O bus (or busses). Digital, during its history, has developed several major busses. These include the MASSBUS, the Q-bus, the Synchronous
Backplane Interconnect (SBI) bus, the UNIBUS, the VAXBI bus, the XMI bus, the Digital Storage Systems Interconnect (DSSI) bus, the Digital Storage Architecture (DSA) bus, and the Laser System Bus (LSB). It is usual that a particular bus might serve as a system bus within a certain processor and as an I/O bus within another processor. For example, the VAXBI bus is the system bus within the VAX 8200, but is used as an I/O bus in the VAX 6000 (where an XMI bus serves as the system bus).

Examples of system configurations for the VAX-11/780, VAX 8200, and DEC 7000 (an Alpha system) are shown in Figures 3.13, 3.14, and 3.15, respectively. As can be seen in Figure 3.13, the SBI serves as the system bus for the VAX 11/780, with a MASSBUS and a UNIBUS serving as I/O busses. Also shown in Figure 3.13 are blocks that identify the location of the CPU, memory, the MASSBUS Adapter (MBA), the UNIBUS Adapter (UBA), and various devices. Each device block represents a combination of a device controller and an associated peripheral device. For the VAX 8200, shown in Figure 3.14, the VAXBI serves as the system bus and a UNIBUS serves as the I/O bus. A block showing the location of a DWBUA adapter is contained within this figure. This adapter allows for communication between the VAXBI and UNIBUS. A DEC 7000 system is shown in Figure 3.15. Here, the LSB is the system bus and an XMI bus serves as an I/O bus. A KFMSB adapter, which allow the XMI bus to interface with a DSSI bus, is also shown. The examples just noted show how major components are connected. There are, however, other possible configurations.

Appendix A, “Evaluated Hardware List (VAX and Alpha),” provides a list of all processors that are included in the evaluated configuration, and identifies the system bus for each processor.

### 3.3.2 I/O Bus Adapters and Controllers

An I/O bus adapter connects an I/O bus to the system bus, thus allowing communication among the I/O bus, the CPU(s), and memory. An MBA and a UBA are identified in Figure 3.13, a DWBUA adapter is identified in Figure 3.14, and XMI and KFMSB adapters are identified in Figure 3.15.

Adapters perform address translation in two different directions: from physical memory to bus address format and vice versa. An adapter also performs device arbitration to handle device requests for the bus. A particular address range in I/O space is reserved for bus adapters. The address range differs depending on the bus to which the adapter is being connected. As an example, the DWMVA adapter is used to connect the I/O segment of the VAX 6000 XMI system bus to the industry-standard Versabus Module Eurocard (VME) bus. This is accomplished by the DWMVA adapter by establishing a handshaking protocol between the two busses, then acting as a channel for data flow between the busses.

A controller is a hardware component that governs activity between the CPU and one or more devices. All Digital devices are connected to a processor through a controller.

### 3.3.3 Console Subsystems

A console subsystem is a collection of hardware, software, and firmware components that provides bootstrap operations and hardware diagnostics for a node (VAX or Alpha). All nodes require a console subsystem. A console subsystem consists of console software and a console terminal. VAX console subsystems allow for an optional console block storage device.\(^8\)

\(^8\) Some console subsystems can support an optional remote diagnostics port. This feature is not allowed in an evaluated configuration.
3.3. VAX OR ALPHA SYSTEM NODE

Figure 3.13. VAX-11/780 System Configuration
Figure 3.14. VAX 8200 System Configuration
Figure 3.15. DEC 7000 System Configuration
CHAPTER 3. HARDWARE ARCHITECTURE

A console subsystem can be thought of as the control panel of the CPU. Some varieties of systems no longer have control panel switches or indicators; they are completely represented and emulated via the console subsystem. The console subsystem associated with a processor has complete and total access to the hardware. Consequently, the console subsystem must be protected from unauthorized access. All console subsystems (VAX and Alpha) are physically part of the base CPU and are protected by the same physical protection given that of the base system hardware.

Console subsystems provide the following standard features:

• Interaction with the processor through the console command language
• Input/Output to the console disk or tape drive (VAX only)
• Loading of diagnostic and bootstrap programs
• Unattended rebooting of the system after a power failure.

A console subsystem operates in console I/O mode or program I/O mode, with one mode always enabled when the processor is running.

3.3.3.1 Console I/O Mode

In console I/O mode, commands entered at the console terminal control the operation of the VAX CPU. The commands are interpreted by the console program and enable the following operations to be performed:

• Initialize the CPU
• Locate physically contiguous memory
• Read data from the console-loading device
• Start the CPU
• Halt the CPU
• Transfer files from the console storage device to CPU memory
• Examine and deposit information in registers or memory locations
• Restart a halted program.

The hardware does not communicate directly with the console subsystem. Information about the state of the hardware is placed in various CPU-dependent registers that are available to the console program when the console is in console I/O mode.

3.3.3.2 Program I/O Mode

In program I/O mode, the console terminal is controlled by OpenVMS/SEVMS, not by the console program. The characters typed at the console terminal are transferred to the processor as terminal I/O. In essence, the console terminal functions like a regular terminal running under OpenVMS/SEVMS when the console subsystem in the program I/O mode, except that the execution of a Control-P sequence at the console terminal changes the console subsystem to console I/O mode.

If a VAX console subsystem has a block storage device, the CONNECT CONSOLE command can be used to make the device accessible from OpenVMS/SEVMS. However, the applicable Trusted Facility Manual
3.3.3.3 Console Program Implementations

Console programs have been implemented on a variety of different console processors, including a Micro-VAX II (e.g., the VAX 8810), an LSI 11/03 microprocessor (e.g., the VAX-11/780), an NVAX+ chip (e.g., the VAX 10000 Model 630), and the VAX CPU itself, with no separate console processor (e.g., VAX 6000 Model 550). In addition to the basic console program, a VAX console subsystem may have built-in firmware diagnostics that can be executed using console commands. For VAXstation systems, the console program supports a "firmware password" that limits the console commands that can be used.⁹ Without knowing the password, a user can enter only non-administrative console commands (e.g., boot from a default device). The TFM describes how an administrator can use this feature to protect the console subsystem in a workstation environment.

For Alpha systems, the console program is implemented in firmware, and AlphaServer 2100 systems have a firmware password feature.

3.3.4 Storage Devices

Digital has developed an architectural approach to storage systems design known as the Digital Storage Architecture (DSA). DSA is an extensive set of standards and specifications that is reflected in all DSA products. Besides specifying aspects of product design such as performance optimizations and error-correction codes, DSA ensures that adding a new Digital storage device to a system will simply involve plugging it in, and will not involve rewriting drivers, the use of protocol converters, or additional adapters.

The DSA standard for connecting large system disk drives and controllers is known as the Standard Disk Interconnect (SDI). SDI specifies a four-line serial connection with each drive connected directly to its controller, and transformer coupling for ease of configuration.

The DSA standard for connecting tape formatters and controllers is known as the Standard Tape Interconnect (STI). STI specifies a four-conductor serial connection, with each drive connected to its controller, and transformer coupling for ease of configuration.

3.3.4.1 Disk Devices

Digital has created five magnetic disk product families, as follows:

- RZ family disks (for SCSI-based workstations)
- RX family flexible diskette drives (for low-end systems)
- RD family disks (for low-end system)
- RF family disks (for low-end and midrange systems)
- RA family disks (for low-end, midrange, and large systems)

⁹ The firmware password feature is available on VAXstation systems having a boot ROM of Version 1.3 or higher.
The five families are distinguished by the storage interconnect used to transfer data between the drive and its controller.

### 3.3.4.2 Tape Devices

All Digital tape storage products implement the DSA and the Digital Tape Mass Storage Control Protocol (TMSCP). Digital tape storage products fall into two basic categories, those for stand-alone processors and those for VMScluster environments. Digital tape storage products for stand-alone processors are complete subsystems with their own built-in intelligent controllers. VAXcluster-based products, however, are attached to a storage server using the STI that allows them to be shared among several VAX computers.

### 3.3.5 Terminals and Printers

Digital offers a wide range of alphanumeric and graphics terminals. Alphanumeric terminals (commonly called text terminals) display letters, numbers, and special characters in a fixed format, while graphics terminals manipulate picture elements on the screen and can also display graphs, charts, and pictures. Graphics terminals also have alphanumeric capabilities.

Digital printers support one or more levels of the Digital ANSI-Compliant Printing Protocol (based on ANSI X3.64), implementing all, or a subset of, the functions defined by that level. Each higher level includes all of the functions defined by the lower levels. Each protocol supports a reset sequence to reinitialize the printer. Each level is briefly discussed below.

- **Printing Protocol Level 1**: Level 1 of the Digital ANSI-Compliant Printing Protocol includes the functions for a basic character cell printer. Functions for backwards vertical movement, vertical margins, or vertical tabs are not included in Level 1.

- **Printing Protocol Level 2**: Level 2 of the Digital ANSI-Compliant Printing Protocol (DEC PPL2), includes the functions for an advanced character cell printer. Functions for limited backwards vertical movement, vertical margins, and vertical tabs are included in this level.

- **Printing Protocol Level 3**: Level 3 of the Digital ANSI-Compliant Printing Protocol (DEC PPL3) supports the DEClaser printers. Functions for proportionally spaced text are included in this level.

### 3.3.6 VAX Processors

Digital offers a wide range of VAX systems that include data center mainframes (e.g., VAX 9000 systems), expandable departmental and data center systems (e.g., VAX 6000 systems), flexible work group and departmental systems (e.g., MicroVAX and VAX 4000 systems), and workstations (e.g., VAXstation systems). This section describes several of the processors analyzed for this evaluation. Because of the large number of processors it was not possible to describe each of them. Selection of specific processors for description and/or comment was made either for historical reasons or because of the popularity of the processor within

\[\text{\underline{10}}\] A basic character cell printer is a monospaced device that may support different horizontal pitches.

\[\text{\underline{11}}\] Advanced character cell printers are monospaced devices that support various horizontal pitches and fonts.
Digital's customer base. The VAX-11/780 is discussed in more depth than the other processors since it was the first member of the VAX family announced by Digital.\footnote{Frequently the performance of a particular VAX processor is compared to the performance of a VAX-11/780. Within such a comparison, the performance of a VAX-11/780 is assumed to be 1.0.}

### 3.3.6.1 VAX-11/780

The VAX-11/780 processor, announced in October 1977, was the first VAX marketed by Digital. A VAX-11/780 is a single processor system that can support up to 100 active users at a given time. As can be seen in Figure 3.13, the CPU, main memory, and I/O subsystem are interconnected via the SBI system bus. The SBI provides a 13.3 Mbits/second path. Traffic on the SBI is reduced by the use of an 8K cache that provides rapid access to frequently used data. A cache is also used to provide high speed address translation. A basic VAX-11/780 system configuration consists of up to 16 Mbytes of main memory, a single UNIBUS, and a console subsystem. The UNIBUS, with a 1.5 Mbits/second path, is used to interface medium and low speed peripherals with the CPU. Optionally, up to four MASSBUS buses can be used to support disk and tape block transfers, with the MASSBUS providing a 2 Mbits/second path. The console subsystem for the VAX-11/780 consists of an LSI-11 microcomputer and a RX01 console load device. The RX01 is used to load system microcode, bootstrap load the operating system, update software, and to support diagnostics.

### 3.3.6.2 Selected MicroVAX, VAXserver, and VAXstation Systems

MicroVAX and VAXserver systems are primarily differentiated by their processing speed and storage capacities. For example, a MicroVAX 3100 exhibits up to 3.5 times the performance of a VAX-11/780 and can support a maximum disk storage capacity of 4.8 Gbytes, as compared to a MicroVAX 3300, which only exhibits up to 2.4 times the performance of a VAX 11/780, but which can support a maximum disk storage capacity of 14 Gbytes. MicroVAX and VAXserver systems use either the Small Computer System Interface (SCSI) bus or the DSSI bus for I/O. The MicroVAX 3300 supports one DSSI bus that provides a maximum I/O throughput of 4 Mbits/s, as contrasted to a MicroVAX 3100 which supports a SCSI bus that provides a maximum I/O throughput of 1.5 Mbits/s.

The VAXstation single-user workstation is based on Digital's chip implementation of the VAX architecture. In addition to support for memory, peripheral devices such as disks and tapes, and Ethernet, VAXstation processors contain connections for a keyboard, mouse, modem (not in an evaluated configuration), a printer/plotter, and a monitor. The VAXstation 4000 Model 60, for example, is a single processor system that exhibits up to 12.0 times the performance of a VAX-11/780. A VAXstation 4000 Model 60 uses SCSI for I/O, with the maximum I/O throughput being 5.0 Mbits/s. The system can support up to 104 Mbytes of memory and a maximum disk storage capacity of 6.8 Gbytes.

### 3.3.6.3 Selected VAX 4000 Systems

VAX 4000 systems included in this evaluation are the Model 300 and the VAXserver Model 300. The VAX 4000 Model 300 exhibits up to eight times the performance of the VAX-11/780. The VAX 4000 Model 300 and the VAXserver 4000 Model 300 are similar to the MicroVAX in architecture. Both are Q-bus based and support up to four DSSI buses for I/O. The VAX 4000 Model 300 can support up to 224 Mbytes of primary
memory. The VAX 4000 and VAXserver 4000 are differentiated only in that the VAXserver 4000 supports disks with larger storage capacities.

3.3.6.4 Selected VAX 6000 Systems

The VAX 6000 Model 510 is a single processor system. Several VAX 6000 models, however, are multiprocessor systems. These include the VAX 6000 Model 620 (two processors) and the VAX 6000 Model 640 (four processors). The VAX 6000 Model 510 exhibits up to 13 times the performance of a VAX-11/780; the VAX 6000 Model 620 up to 58 times the performance of a VAX-11/780; and the VAX 6000 Model 640 up to 106 times the performance of a VAX-11/780. For VAX 6000 Models 400, 500, and 600 systems, the maximum I/O throughput is 80 Mbits/s. VAX 6000 Model 510 can support up to 512 Mbytes of memory. Numerous I/O busses are used within VAX 6000 systems. As an example, a VAX 6000 Model 210 can employ, for I/O, one XMM bus, four VAXBI busses, and eight CI busses. VAX 6000 systems support vector processing.

3.3.6.5 Selected VAX 9000 Systems

VAX 9000 systems range from a single processor system (e.g., the VAX 9000 Model 210) to a variety of multiprocessor systems (e.g., the VAX 9000 Model 420 (two processors); the VAX 9000 Model 430 (three processors); the VAX 9000 Model 440 (four processors)). A VAX 9000 Model 210 exhibits up to 40 times the performance of a VAX-11/780 and a VAX 9000 Model 440 exhibits up to 157 times the performance of a VAX-11/780. Maximum I/O throughput for a VAX 9000 Model 210 is 80 Mbits/s, and that for a VAX 9000 Model 440 is up to 320 Mbits/s. VAX 9000 systems can support in excess of 7 Tbytes of disk storage capability. Several types of I/O busses are found within VAX 9000 systems. For example, a VAX 9000 Model 440 can support 4 XMM I/O busses; 10 CI I/O busses; and 14 VAXBI I/O busses. VAX 9000 systems support vector processing.

3.3.6.6 Selected VAX 7000 and VAX 10000 Systems

VAX 7000 and VAX 10000 systems, also known as Laser/NVAX+ systems, are a collection of Digital’s newest processors. These systems are based on the NVAX+ CPU chip and include the VAX 7000 series, Models 610, 620, 630, and 640, and the VAX 10000 series, Models 610, 620, 630, and 640. Laser/NVAX+ systems communicate with main memory via the Laser System Bus (LSB), and can support from one to four CPUs, from 64 Mbytes to 2 Gbytes of primary memory, and from one to four XMM busses for I/O. A unique feature of Laser/NVAX+ systems is that they employ mailbox I/O rather than memory mapped I/O. A Laser/NVAX+ processor includes one Laser I/O Port Module (IOP) that interfaces up to four separate I/O busses to the LSB.

3.3.7 Alpha Processors

For the RAMP Action that is addressed by this Final Evaluation Report (FER), Digital selected three Alpha series systems for evaluation, the DEC 7000 series, the DEC 10000 series, and the AlphaServer 2100 series. A brief description of each follows.

FINAL: 24 October 1996
3.3.7.1 DEC 7000 and DEC 10000 Systems

DEC 7000 and DEC 10000 systems, also known as Laser systems, are a collection of Digital’s newest processors. These systems are based on the DEChip 21064 and DEChip 21064A Alpha CPU chips. Model numbers include the DEC 7000 series, Models 610, 620, 630, and 640, and the DEC 10000 series, Models 610, 620, 630, and 640. Laser systems communicate with main memory via the LSB, and can support from one to four CPUs, from 64 Mbytes to 2 Gbytes of primary memory, and from one to four XMI busses for I/O. A unique feature of Laser systems is that they employ mailbox I/O rather than memory mapped I/O. A Laser processor includes one Laser I/O Port Module (IOP) that interfaces up to four separate I/O busses to the LSB. The difference between the DEC 7000 and DEC 10000 is the I/O capacity (i.e., the number of I/O buses that can be connected).

3.3.7.2 Alpha 2100 Systems

AlphaServer 2100 systems are low-cost symmetric processing (SMP) PCI/EISA-based servers that are suitable for general-purpose commercial high-performance applications. These systems are based on the DEChip 21064 and DEChip 21064A Alpha CPU chips. Each AlphaServer system can be configured with up to four processors of the same speed for symmetric multiprocessing. The systems support up to 2GB of memory and 64GB of internal disk storage. The system bus bandwidth is 667MB/second and the high-performance PCI I/O subsystem has a peak bandwidth of 132 MB/second. The 33 MB/second EISA I/O bus supports a variety of industry-standard EISA options.

3.4 VMScluster

The following sections discuss the components of the VMScluster architecture. A general discussion of cluster configurations is followed by descriptions of the Computer Interconnect (CI) Cluster, the Network Interconnect (NI) Cluster, and the Mixed Interconnect Mixed-Interconnect (MI) Cluster. A VMScluster configuration may contain VAX and Alpha nodes; this is known as a Mixed Architecture VMScluster (MAVC) configuration.

3.4.1 Configurations

There are four basic hardware configurations defined for the OpenVMS/SEVMS evaluated configuration. They are as follows:

1. A standalone system (any single VAX node or any single Alpha node listed on page 227, “Processors”).

2. A CI VMScluster. Such a VMScluster can be either “single architecture” (all VAX nodes or all Alpha nodes) or “mixed architecture” (VAX and Alpha nodes).

3. An NI VMScluster. Such a VMScluster can be either “single architecture” (all VAX nodes or all Alpha nodes) or “mixed architecture” (VAX and Alpha nodes).
4. An MI VMCluster combining the configurations mentioned in 2 and 3. Such a VMCluster can be either “single architecture” (all VAX nodes or all Alpha nodes) or mixed architecture (VAX and Alpha nodes).

For each of the configurations identified above there cannot be connections to systems outside of a uniform security domain. Therefore, an appropriate amount of physical security is required for physical interconnections.\textsuperscript{13}

\section*{3.4.2 Stand-alone System}

The stand-alone system can be thought of as any single VAX node or any single Alpha node running unattached to any other. A stand-alone system does not have to be a VMCluster.

\section*{3.4.3 The CI Cluster}

Figure 3.16 depicts an example of a single architecture CI VMCluster and Figure 3.17 depicts an example of a mixed architecture CI VMCluster. The key components of the CI VMCluster are the processors, the CI bus, the Hierarchical Storage Controller (HSC), and the Star Coupler. The CI bus is a dual-path bus that, along with the CI adapter, provides the physical connection from each cluster node to the hub, known as the Star Coupler. The dual paths in the CI bus exist for redundancy in case of path failure. When both paths are available, they can be used in parallel. The CI adapter is a communication port; it performs data buffering, address translation, and encoding and decoding all under microprocessor control.

Multiple CI interfaces can be configured for each node using extensions to the CI architecture. Currently, OpenVMS/SEVMS supports up to four CI interfaces per node and two star couplers per VMCluster.

\subsection*{3.4.3.1 HSC}

The Hierarchical Storage Controller is an intelligent storage device controller that manages and controls reading and writing of information from disks and tapes in a cluster. An HSC is considered a cluster node; however, an HSC is passive and does not initiate any requests to other nodes. It simply responds to I/O requests from VAX or Alpha nodes. It is a part of the Trusted Computing Base (TCB) and is not directly accessible to untrusted subjects. In addition, no access rights are checked by an HSC. All access checks are made in VAX or Alpha nodes.

There are seven models of HSC in the evaluated configuration. The HSC40, the most basic model, supports up to 12 disk or tape ports. The most powerful model, the HSC95, supports up to 48 devices, up to 24 of which may be tape drives. The HSC95 also contains 64 MB of cache memory that provides a performance enhancement (currently, only 32 MB of cache are supported by the HSC software).

The interface to each device is called a data channel, which is itself controlled by a microprocessor. Each data channel can support up to four disks or four tape formatters. The data channels schedule and perform the read, write, seek, and status operations to the devices.

\textsuperscript{13}The applicable OpenVMS/SEVMS TFM warns the administrator to physically protect the console terminals and network devices (such as cables and repeaters), and not to use modems or remote diagnostic capabilities.
Processors must be all VAX or all Alpha

Figure 3.16. A single architecture CI VMScluster
Figure 3.17. A mixed architecture CI VMScruelster
The primary hardware components of the HSC are the console, the control RAM, the internal control bus, the data RAM, the data bus (also internal), the host interface microprocessor, the control microprocessor, and the data channel microprocessors. The host interface microprocessor is directly connected to the CI bus. HSC memory is used for three distinct purposes: program, control, and data. Program memory contains software; control memory holds data (control) structures; and data memory contains 512 KB of buffers.

The HSC runs a total of 125,000 lines of microcode and software. Because the control microprocessor has exclusive use of the program memory, all HSC software is executed by the control microprocessor. The other microprocessors execute only microcode. The internal HSC processors communicate through work queues held in control memory containing I/O commands to be executed on behalf of the VAX or Alpha processors. The interface microprocessor executes microcode that uses a polling algorithm to fetch “work packets” from the queues in control memory. The execution of the packets is initiated; then the interface processor monitors the progress of the packet, waiting for it to finish.

With the exception of the HSC50, HSC65, and HSC95, all of the other models run the same software. The HSC65 and HSC95 run an upgraded version of the HSC software that includes support for larger amounts of cache memory (relative to the HSC60 and HSC90 models) and improved caching algorithms. The HSC 50 runs an earlier version of the HSC software.

The control microprocessor runs software that can be thought of as a mini-operating system that performs multitasking. There is an executive that schedules processes that execute I/O tasks and utilities. The following is a list of the entities that execute on the control microprocessor (invoked from the HSC console) and a one-line description of their functions:

- EXEC - Scheduler
- DISK - Disk I/O command executor
- TAPE - Tape I/O command executor
- ECC - Disk error recovery process
- VTDPY - HSC operation display process
- BACKUP - Disk-to-tape backup utility (not the same as the OpenVMS/SEVMS utility)
- VERIFY - SDI disk drive verification utility
- DEMON/PDEMON - Diagnostic execution monitors
- DUP - Diagnostic and Utility Protocol Server
- POLLER - CI node polling process
- SCSDIR - Process that provides information to cluster nodes about available server processes on the local HSC
- SETS$HO - Utility to manipulate HSC information

The SETS$HO utility program allows an operator to view or alter HSC operational parameters. The functions of SETS$HO include the following: defining the HSC name, changing error reporting and diagnostic execution parameters, setting tape support capacity, and enabling or disabling connections with host computers. Some
of the SETSHO commands force a reboot of the HSC (e.g., tape support). Digital advises that certain SETSHO commands not be executed while there is active I/O on the HSC.

A boot device, located inside the front cabinet, is used to load the operating system for the control processor during powerup or initialization of the HSC. The boot device for the HSC40 and HSC70 is a flexible disk drive; for the HSC50, it is a cassette drive. The boot device is located inside the front cabinet door which is lockable.

Also inside the HSC front cabinet, there is a switch that can be set to one of two positions labeled “Secure” and “Enable.” The switch enables or disables the pushbuttons on the operator control panel, located on the outside of the cabinet. The switch also affects software that may be run on the HSC. When the switch is set to the Secure position, no SET commands may be run on the HSC node. In addition, the break key, which allows an operator to enter commands on the HSC console terminal, is disabled. The SHOW commands may be run when the switch is set in either position.

The operator control panel contains three buttons for initializing the HSC, displaying fault information (if a fault has occurred), or configuring the HSC to be on or off line. When the HSC is on line, cluster communications are allowed. When the HSC is switched off line, no new cluster communications are allowed to be established. Existing communications are allowed to complete normally.

The pushbuttons are accessible even when the cabinet door is closed. When the switch is set in the Secure position, pushing the control panel buttons has no effect on the HSC. Tampering with the control panel is prevented by setting the switch to the Secure position and locking the front cabinet.

The Mass Storage Control Protocol (MSCP) is used in the cluster to communicate with the HSC controllers over the CI bus. An MSCP command is part of a sequenced message sent to the HSC via the CI bus. The MSCP commands are moved into the control memory work queues by the interface processor. In the case of disk I/O, the DISK process running on the control microprocessor converts the MSCP command to SDI command(s) to be executed by the data channel. The DISK process is also responsible for placing seek commands in the target drive’s seek queue to optimize performance. The TAPE process performs the analogous functions for tape I/O.

The HSC also supports a volume shadowing capability. The operator can command the HSC to treat a set of disks as duplicates of another set. To the user/application, the volumes appear as one set. Volume shadowing is not strictly a hardware capability for HSC disks. For the HSC, write commands are performed on all the disks in the set before completion status is reported. For reads, the HSC will optimally select a drive to which to issue the I/O command. Shadowing sets can be reconfigured; volumes can either be removed or added under OpenVMS/SEVMS control while the HSC is online. Disks not affected by the reconfiguration procedure remain available for use.

3.4.3.2 Star Coupler

The Star Coupler is the central, physical connection point of a CI cluster. Its function is to electronically isolate each data line and to distribute data to and from cluster nodes. The Star Coupler is a passive entity; it merely sends out what it receives. It initiates no action on the cluster. The unit contains up to four panels to which coaxial cables (the CI busses from cluster nodes) are connected. The basic Star Coupler can support a maximum of 16 nodes. With a Star Coupler Expander, a Star Coupler can support up to 32

---

14 The HSC console must also be physically protected, so that only operators and administrators have access to the console, as stated in the TFM.
CI-nodes. However, there is a limit of 16 processors per CI. Nodes can be added or removed one at a time without interrupting other nodes. The maximum length of a coaxial cable connection to the Star Coupler is 45 meters.

### 3.4.3.3 DSSI Bus

The Digital Storage System Interconnect (DSSI) is a bus that serves a similar purpose as the CI and HSC configuration in that it facilitates disk usage. The bus is essentially an I/O bus that uses the MSCP protocol. The DSSI bus provides fewer functions than those of the CI and of the HSC. The bus cannot be used as a substitute for the CI. That is, a DSSI can be used to connect multiple hosts to multiple disks, but that configuration by itself is not considered a cluster.

Only certain storage devices (disks) can be hooked to the DSSI. This is because certain HSC features are integrated with the device that result in the device performing its own controller functions. For example, a DSSI device can handle communication with more than one host. MSCP is implemented in the microcode on the DSSI disk drive; the physical placement of the data on the disk is handled by the device itself. Command optimization is also performed by the device. In addition, a DUP server, similar to the HSC DUP server, is implemented in the microcode in the device.

Two HSC features not supported in the DSSI architecture are volume shadowing and internal backup functions. Backup for DSSI disks is performed from a VAX or Alpha node.

The DSSI bus supports up to eight connections, seven of which can be storage devices (disks). The maximum length of the bus is 12 meters.

### 3.4.4 The NI VMScluster

Figure 3.18 shows a sample configuration of a single architecture NI VMScluster, and a sample mixed architecture NI VMScluster is displayed in Figure 3.19. The most obvious difference between an NI and a CI cluster is the communication medium between nodes. Ethernet and the Fiber Distributed Data Interface (FDDI) are used in an NI cluster instead of a CI bus.

Each cluster can be divided into a maximum of three segments (either Ethernet segments or FDDI rings) for the purposes of providing more reliability and enhancing cluster performance. For a segmented cluster, Local Area Network (LAN) adapters and bridges are used to connect the segments into one cluster. Open VMS/SE-VMS can be configured to survive a bridge failure through the SYSGEN utility.

Cluster nodes are made up of one or more boot servers and satellite nodes. Boot servers are disk servers that download executable images over the busses to other NI nodes. Satellite nodes are VAX or Alpha processors that are booted over the busses by a boot server. Not all processors can be configured as satellite nodes. Only the VAX 3000 and 4000 series, MicroVAX II, VAXstation II, and VAXserver (with the exception of the VAXserver 6000 series) or AlphaServer 2100 series may be set up as satellites. An NI cluster boot server must be one of the following processors: VAX 6000 series, 7000 series, 86xx, 86xx, 8700, 88xx, 9000 series or 10000 series. For Alpha, DEC 7000 series, DEC 10000 series or AlphaServer 2100 are included. In addition, since HSC controllers and Star Couplers require CI bus connections, they cannot be configured into an NI. Boot servers are usually the most powerful machines in the cluster for reasons that are explained below. Digital recommends that boot servers and system disk servers be connected to the same LAN segment and
that all cluster nodes have a direct path to other nodes.

Boot servers provide both cluster and resource management. The system disk contains common cluster files for startup, authorization, directories from which the satellites are booted, and other system table configuration information. Examples of resources provided by the boot server include user/application data disks, printers, and distributed batch-processing capabilities.

The satellite nodes are booted using the DECnet Maintenance Operations Protocol (MOP). During the response to a boot request from a satellite node, a boot server sends an image to the satellite. This image facilitates loading of OpenVMS/SEVMS on the satellite and enables it to become a member of the cluster (see page 130, “System Startup”, for more details on cluster boot procedures). Satellite nodes can also perform disk serving and batch-processing functions. They are often configured with local page and swapping file disks.

3.4.4.1 Ethernet

A baseband (as opposed to broadband) Ethernet configuration is used in an NI cluster. In a baseband configuration, a single channel on the cable exists and is only used for Ethernet communications. The hardware components of the basic Ethernet connection are the communication controllers, the transceivers, and the repeaters, and the coaxial cables. None of the Ethernet components contains any security functions. The hardware is protected by physical security. Figure 3.20 shows a basic Ethernet connection. Ethernet connections may also contain a Device Local Network Interconnect (DELNI) unit that is described below.

Communications Controllers  The Ethernet communications controllers are intelligent devices that directly connect to the backplane of a VAX or Alpha processor. The controllers provide the physical and data link layers of the network protocol. The actual model of the controller used varies, but is particular to the bus structure (system bus) of the processor. Each controller supports one Ethernet port. The communication functions performed by the hardware in the CI cluster (i.e., the CI port interface) are emulated by the software that interfaces to the Ethernet controller. Thus, the higher level software (the System Communication Services (SCS) routines) used to communicate over either the CI or the NI cluster is the same.

Transceivers  The Ethernet transceivers contain the interface hardware that connects an NI node to the Ethernet coaxial cable. The transceivers are the Ethernet “taps.” The transceivers transmit to and receive signals from the cable. In addition, the transceivers detect message collisions. In a basic Ethernet connection, a transceiver directly connects to a communication controller via a cable that may be up to 50 meters in length. As in the case of the controllers, the actual model of transceiver used depends on the bus structure of the processor to which it connects.

The transceiver contains a logic module that performs its functions. Each model also contains some redundant protective circuitry and self-testing functions. There is also a continuous message loopback feature that supports fault isolation. The transceiver also provides cable signal noise immunity.

Repeaters  The Ethernet repeaters, local and remote, effectively extend the length of a network connection. Each repeater connects, via coaxial cable segments, two transceivers. Each cable segment can contain up to 99 transceivers. The local repeater can connect cable segments that are up to 100 feet apart; the remote
Boot server and satellites must be all VAX or all Alpha.

Figure 3.18. A single architecture NI VMSCluster
Figure 3.19. A mixed architecture NI VMScluster
Figure 3.20. Basic Ethernet Connection
repeater can connect cable segments up to 1000 feet apart. The remote repeater is actually two local repeaters linked together by an fiber optic cable. There is a fiber optic logic interface module that can be added to a local repeater to alter it to be a remote repeater. There is also a multiple port repeater used with the VAXstation. The multiple port repeater can connect up to eight Ethernet cable segments and connects to one transceiver. All of the repeaters perform the same function. They take the input signal from a transceiver and re-time, reamplify, and repeat the signal to the rest of the network over a single coaxial cable in the case of the local or remote repeater or over multiple lines in the case of the multiple port repeater. Repeaters have self-test and fault signal detection capabilities.

**Local Area Network Interconnect (DELNI)** The Local Area Network Interconnect is a device that allows up to eight NI nodes to be connected via the DELNI interconnect to one transceiver. With a DELNI configured into an NI cluster, the number of transceivers and the length of cable required to construct the network is reduced. Figure 3.21 shows an NI cluster using a DELNI. DELNI interconnects can also be hooked directly into an Ethernet coaxial cable (in a stand-alone configuration) and connected to CI nodes.

### 3.4.4.2 FDDI

Some processors, namely the VAX 6000 series, 7000 series, 9000 series, 10000 series and the Alpha DEC 7000 series, 10000 series and AlphaServer 2100 series support the FDDI. The FDDI is a token ring protocol. The bus has a basic speed of 125 megabits per second. The Ethernet driver also supports the FDDI. The adapter used in the connection with the fiber optic bus is called the DEMFA. As in the case of the Ethernet, security of the bus is maintained through physical means.

### 3.4.5 Mixed Interconnect VMScluster

The mixed interconnect MI VMScluster configuration combines both the CI VMScluster and NI VMScluster. Figures 3.22 and 3.23 show examples of a single and mixed architecture MI clusters, respectively. Note that one CI cluster is combined with one NI cluster. The number and type of processors, HSC controllers, and Ethernet-related devices may vary. The maximum number of Star Couplers that can be used is restricted to two, for reliability purposes only. The mixed interconnect cluster allows a satellite node to access disks managed through a CI connected node.
3.4. VMSCluster

![Diagram of DELNI Connection]

Figure 3.21. DELNI Connection
Processors must be all VAX or all Alpha.

Figure 3.22. A single architecture MI VMScluster
Figure 3.23. A mixed architecture MI VMScluster
This page intentionally left blank
Chapter 4

Software Architecture

This chapter describes the major software components of the OpenVMS/Security Enhanced VMS (SEVMS) Trusted Computing Base (TCB). The first section is a description of the overall software structure of the TCB. It is followed by a discussion of the mechanisms used by the TCB for communication among its internal components both within a Virtual Address eXtension (VAX) or Alpha node and between the nodes of a cluster. Sections on OpenVMS/SEVMS memory management, process management, interprocess communication mechanisms, file management, and input/output management follow. The chapter ends with a discussion of system operation, startup, shutdown, and installation.

4.1 Software Structure

The software of OpenVMS/SEVMS is composed of a number of images. An image is a collection of program code and/or data. An image may be a complete executable program, or may rely on other images. Images are described more fully starting on page 90, “Images”. A “known image” is an image that has been “installed” by specifying information about how to execute the image. Security-relevant information that may be specified includes extra OpenVMS/SEVMS privileges (see page 138, “Privileges”) that are granted to a process while executing the image. An image may also be installed with the ability to run in the protected modes (kernel, executive, and supervisor) of the processor. Known images installed with any of these security relevant attributes permit a process to perform operations that would otherwise not be available to untrusted processes while constraining the operations to maintain the system security policy. The additional abilities granted to the process are limited to the duration of the known image’s execution as described on page 100, “Known Images”.

TCB images may either be system images that provide services to every process on the system, or process images that provide operations locally to each process using them. Each of these categories will be further refined and subdivided for detailed description.

4.1.1 Executive Images

Executive images exist in VAX nodes, Alpha nodes, and Hierarchical Storage Controller (HSC) nodes. In a VAX node, a number of images are combined to form the VMS executive. These images run in the protected modes of the hardware. In the case of an HSC node or the system console, all software running directly on the HSC or system console is part of the TCB and not part of any OpenVMS/SEVMS process.

4.1.1.1 VMS Executive

The term VMS executive refers to those parts of OpenVMS/SEVMS that reside in the system virtual (S0)
address space. S0 address space is intended for use by system code, but is mapped into every process's virtual address space and can be addressed by unprivileged processes. The VMS executive is protected by setting the hardware memory page protections to disallow access by instructions being executed in the user mode of the processor. All S0 pages are protected against user mode write; and, in general, pages containing system data structures are not accessible at all from the user mode of the processor.

The core of the VMS executive is made up of the base image and approximately 20 separately loadable images. The base image contains transfer vectors to routines in the loadable images and storage for system global variables. A loadable executive image consists of modules performing related functions and data and initialization code specific to those functions. An example of this is the process management image that includes the scheduler and process creation and deletion routines.

As each loadable image is loaded into system space, its associated transfer vectors in the base image are modified to contain the addresses of its routines. Thus, the base image transfer vectors can be used by all the loadable images to transfer control to another module. Initially, all the pointers in the base image are set to system error routines within the base image itself. The initialization module of each image is called by the loading routine. This initialization is responsible for setting the page protections for each of the pages of the image.

A number of the loadable images are loaded on a configuration dependent basis. In particular, some images are loaded only if cluster communication hardware is present. Other images contain instruction set emulation code and are loaded if the VAX processor model on which the code is being loaded does not support the entire instruction set. Every processor type has a processor-specific image that contains routines with code specific to that CPU type. An example of the routines in that image would be a routine to clear the I/O bus adapter.

Loadable images are read into system space in several phases during the boot process. These phases are described in the boot process description on page 130, “System Startup”, and the loadable images are listed on page 243, “VMS Executive Images”.

The VMS executive also contains device driver code and a number of system data structures built during initialization. An administrator may add devices to the system I/O databases while the system is operational. However, the system must be rebooted to remove most types of devices from the I/O databases.

4.1.1.2 VAX Console Subsystem

The console subsystems associated with each VAX node are described on page 48, “Console Subsystems”. These consoles vary from processor to processor. A few are strictly hardware implementations; however, many of the console subsystems include software. This software is not accessible from untrusted interfaces and is conceptually part of the VMS executive although it may not execute on the VAX processor itself.

4.1.1.3 HSC Executive

The HSC software is conceptually also part of the executive. It does not execute as an OpenVMS process, but rather as the real-time control program for the HSC. All software running on any HSC is fully within the TCB. The software is not accessible to users, and simply controls the management of disk and tape devices including scheduling I/O requests and dictating read/write head positioning. The software may be used to monitor performance and failures, backup media, and to maintain shadow volumes (see page 58, “HSC”).
4.1.2 Non-Executive Trusted Code

There are three basic methods for an image not in the executive to execute as part of the TCB: (1) an image may be run in a trusted process (a process with OpenVMS/SEVMS privileges) created by the system specifically for that purpose; (2) other images run with privilege in untrusted processes because they have been administratively “installed” as a “known image;” and (3) administrators may run images that utilize privileges granted to the administrator. Each of these methods is discussed below.

4.1.2.1 Trusted Processes

Some processes are created with privilege by other components of the TCB. These processes are used to execute one or more images as a TCB process. Often these are server processes used to manage some component of the TCB. Typical of the OpenVMS/SEVMS privileges used by these processes are CMKRNL and CMEXEC. These privileges allow the process to execute code in processor modes reserved to the system. These processes are trusted for the life of the process. The privileges they possess are not dependent on the image being executed. Some of these trusted processes are started at system initialization. When DEChet is started (typically also at system initialization) many of the network server processes are created. A number of these processes are created the first time they are needed. Unless otherwise noted, all of these processes run with a system Discretionary Access Control (DAC) authorization permitting them to read and modify TCB files.

The Startup Process is one of the first few processes started during system initialization. It runs with all privileges enabled and is responsible for starting many of the other trusted processes. Its execution as part of system initialization is described on page 130, “System Startup”. It is not present after the TCB reaches a secure state.

The Operator Communications Process controls operator requests and replies. It is started by the startup process. It requires CMKRNL, EXQUOTA, NETMBX, OPER, SYSPR, SETPR, and WORLD privileges. In addition, Alpha requires the SYSLCK privilege.

The Swapper is discussed on page 91, “Swapper”, and runs exclusively in kernel mode. This is possible because the swapper process context is designed into the executive. It is not created by another process.

The Audit Server is responsible for maintaining the flow of audit records to magnetic storage and operator consoles. It is discussed in detail on page 163, “The Audit Server”.

The Security Server is responsible for performing operations on the proxy and intrusion databases.

The Magnetic Tape ACPs handle file-based operations to magnetic tape drives. Ancillary Control Processes (ACPs) are discussed on page 126, “Ancillary Control Processes”. These processes run in the user and kernel modes of the processor. They run with all privileges enabled and with a user account in the
system group. SEVMS uses two separate ACPs, one that manages labeled tapes, and a second that manages unlabeled tapes for system administrator use.

The Network ACP controls the lower level link protocols for network communication, and is discussed on page 85, “Network ACP”. This process runs in kernel mode, although it makes file accesses in user mode.

The Remote Terminal ACP creates a pseudodevice in response to a user request on a remote VMScluster node, and deletes it after the the user session ends. Control is passed to a device driver to manage the actual session. This is discussed in the context of DECnet at page 86, “The Remote Terminal Application”. This process runs in the user and kernel modes of the processor.

The Network Event Logger logs significant network errors (i.e., circuit failures) to system logfiles. This process runs with SYSNAM, OPER, and SYSPRV privileges.

The Error Log Format Process collects hardware error information and writes it to the system error log. This process requires BYPASS, CMKRNL, and WORLD privileges. The error log format process is started by the startup process at system initialization time.

The IPC ACP manages communication via the $IPC system service that is routed over a DECnet connection between two nodes in a VMScluster. It runs with CMKRNL, SYSNAM, SYSPRV, TMPMBX, NETMBX, and OPER privileges.

The Job Controller starts all interactive and batch processes. It is created by the startup process and executes with the SETPRV privilege. For additional information on process creation see page 98, “Process Management”.

The Card Reader Input Symbiont is discussed on page 103, “Batch Job Processing”. For VAX, it is created by the job controller whenever the card reader is used to enter a job. It runs with the SETPRV privilege. Card readers are not supported on Alpha.

The Print Symbionts control the printing of files on output devices. In SEVMS the secure print symbiont adds page markings. Symbionts are discussed on page 127, “Queue Management”.

The MSCP Server is created during initialization of a cluster to manage I/O from other cluster nodes to a local node’s cluster visible disk devices.

The System Management Server is responsible for executing distributed commands from the SYSAM utility on VMScluster nodes. It is discussed more extensively on page 83, “The System Management Server”.

FINAL: 24 October 1996
The Device Configuration Manager configures Mass Storage Control Protocol (MSCP) served devices into a node’s I/O databases. It is started only if the node is a member of a VMScluster or has Computer Interconnect (CI) hardware. It requires CMKRN1, PRMMBX, BYPASS, and SHARE privileges.

The Cache Server flushes the system-wide caches and runs with all privileges enabled. It is started only if the node is a VMScluster member.

The Cluster Server acts as an envelope for cluster jobs and is described on page 83, “The Cluster Server”. It runs with all privileges enabled and is started only if the node is a VMScluster member.

The Satellite Loader is created to respond to satellite requests for executable boot code.

The Shadow Server is created to manage shadowed disks. It runs with CMKRN1 and OPER privileges.

The Transaction Processing Server is created during initialization to manage transaction processing. It runs with CMKRN1, SYSNAM, SYSPRV, TMPMBX, NETMBX, and OPER privileges.

The RMS Recovery Server is created by Record Management Services (RMS) to recover journalled files.

The Queue Manager controls the batch and print queues. It is described on page 127, “Queue Management”.

4.1.2.2 Installed Images

Some of the informational utilities and disk and tape volume manipulation utilities require that selected portions of protected data structures be read or written in a controlled fashion. OpenVMS/SEVMS has two methods to facilitate this. Both of these methods require a system administrator to “install” the image that requires this type of access. The first method is to install an image with privilege; the other is to install an image containing a collection of subroutines as a “protected” image.

Complete programs that require privilege to perform their functions are installed (as described on page 100, “Known Images”) by the system manager so that they can perform their functions in an ordinarily untrusted process. For an image installed with privilege, image privileges are acquired upon image activation and removed upon image deactivation as described on page 138, “Privileges”. An example of this type of image is SETP0, the image that allows a user to change entries in the password database.

Protected sharable images are installed with the “protected” option. This does not grant the image any explicit privileges, but instead causes the image to be treated as a collection of system services as described on page 79, “System Services”. This causes the image to be run in a privileged mode of the processor when it is invoked. This mechanism is used to provide what is essentially an extension to the system call interface. The install utility marks all the pages of the privileged image to prevent access by code running in the user mode of the processor.
The images that are installed by default in OpenVMS/SEVMS are listed on page 247, “Images Installed with Security Relevant Privileges”. The Trusted Facility Manual (TFM) provides warnings that any additional images installed with privileges or as protected are not within the OpenVMS/SEVMS evaluated configuration.

### 4.1.2.3 Trusted Users

Administrators of OpenVMS/SEVMS can be assigned privileges by username. Software run in a process started on behalf of an administrator can take advantage of those privileges. The administrator is expected to login to a privileged user account and run only software that is trusted not to abuse privilege. This includes software that does not require privilege to perform its task, but is trusted not to take improper advantage of an administrator’s privileges. The software that an administrator would use to establish and maintain a secure environment is therefore part of the TCB. A specific example of this is the system dump analyzer, which depends on the privileges in an administrator’s process to allow it to analyze the state of the running system. An administrator may also be granted discretionary access to internal TCB data by virtue of having DAC access to TCB data files. The SYSPRV privilege would allow this access to an administrator who was not otherwise permitted DAC access.

### 4.1.3 Security Database

A number of files are critical to the secure operation of OpenVMS/SEVMS. These comprise the security database of OpenVMS/SEVMS. For the evaluated configuration, all nodes must share the same physical copy of the security database files in the shared file system. The security databases may be maintained on shadowed disk volumes. This permits multiple copies of the security database files to exist, but the TCB by utilizing the distributed lock manager will ensure that the copies remain consistent as described on page 124, “Volume Shadowing Driver”. All of the security database files require either privilege or special DAC access rights to access. The files utilized by both OpenVMS and SEVMS are:

- **AUDIT_SERVER.DAT**: Contains configuration data for the audit server. This is maintained by the SET AUDIT command.
- **CLUSTER_AUTHORIZE.DAT**: Contains the NI cluster number and password.
- **NETOBJECT.DAT**: Contains the definitions of the DECnet applications.
- **NETPROXY.DAT**: Contains user authorizations for inter-node use. This is maintained by the AUTHORIZE utility and system services.
- **RIGHTSIVIST.DAT**: Contains identifier database. This is maintained by the AUTHORIZE utility and system services.
- **SYSAZ.L.DAT**: Contains the automatic login facility data base. Maintained by SYSMAN utility.
- **SYSUAF.DAT**: Contains the authorized user database. Maintained by the AUTHORIZE utility and system services.
- **VMS$OBJECT.DAT**: Contains the object protection defaults.
The additional files required by SEVMS are:

SEVMS$LOGIN_CLASS.DAT Contains login classification ranges of terminals; this is maintained by
the SET CLASS /TERMINAL command.

SEVMS$SMB_HDRFRM.DAT Contains the template versus category data base for print symbiont.
Maintained by the SET TEMPLATE command

SEVMS$SMB_LIB.TLB Contains a library of templates for the secure print symbiont to use. Maintained
by the LIBRARY utility.

4.1.4 Untrusted Software Interface

The TCB interface presented to untrusted software is a combination of two basic mechanisms. The Change
Mode to Executive (CHME) and Change Mode to Kernel (CHMK) instructions provide a mechanism for
untrusted code to call specific entry points of the TCB, called “system services”, that run in the protected
modes of the VAX or Alpha processor. The set of privileged executable images on the system provide another
interface. These interfaces are consistent across the VMScluster system even though the TCB executes on
all the VMScluster nodes.

4.1.4.1 System Services

System services are provided to manage memory, processes, and Interprocess Communication (IPC), and
provide security interfaces to the TCB. RMS provides file management services through a similar mechanism.
In general, RMS uses public interfaces to lower layers of the TCB (see page 118, “Record Management
Services”). The same mechanisms that are used to transfer control to system services are also used to
transfer control to protected images (see page 100, “Known Images”). A process invokes a system service
by executing code that contains a CHME or CHMK instruction with a predefined operand. Operand values
between 0 and 255 are used for services loaded into the VMS executive. A process invokes a protected
image by executing a CHME or CHMK instruction with an operand either negative or greater than 255.
The CHME and CHMK instructions cause an exception which causes control to be transferred through the
System Control Block (SCB) (as described on page 24, “System Control Block”) to an OpenVMS/SEVMS
change mode handler. The processor mode is changed to the requested mode during the transfer. The change
mode handler builds some system data structures. If the requested service is part of the executive, it is listed
in the change mode handler and control is transferred to that service. If the requested service is not in the
executive, a protected vector list maintained by the image activator in the invoking process’s P1 address
space is checked. Control is transferred to each routine in the protected vector list. Each routine checks
the operand, and returns success or failure in handling that operand value. Each routine is called until one
successfully handles the operand or all have returned failure. Each system service validates its parameters
by using the VAX probe instructions. The probe instructions access the parameter using only the access of
the caller. The parameter values are copied into system data structures if changes to the values could be
security-relevant. All returns from either type of system service return to a common exit path that returns
the processor state to that of the caller, handles any errors, and then executes a Return from Exception or
Interrupt (REI) to return to the caller.
4.1.4.2 Installed Images

Both images installed with privilege and protected images may be executed by untrusted processes. These images are part of the TCB and form an additional interface to the TCB. Images installed with privilege use the privileges that have been associated with them to perform otherwise restricted operations in well-defined ways. Protected images use their ability to execute code in the protected modes of the processor to perform otherwise restricted operations in well-defined ways. Both types of image are responsible for enforcing the security policy at their interface to untrusted processes. A small number of these images are available to administrators only, and simply exit when invoked without a required privilege. An example of an image that is available for use by untrusted processes is the SETP0 image, which is installed with privilege and allows users to modify their own passwords. Another example is the MOUNTSHR protected image that allows users to mount volumes for which they have DAC; and, in SEVMS, Mandatory Access Control (MAC) authorization.

4.1.5 Privileged Interfaces

Several interfaces to OpenVMS/SEVMS are intended to be used by administrators only. Each VAX, Alpha, and HSC node has a console that is able to perform diagnostic and maintenance functions without any user authentication. All the TCB hardware must be protected from tampering, but the system consoles must also be physically protected from use by unprivileged users. For a discussion of the capabilities of the consoles, see page 48, “Console Subsystems”.

4.2 Internal TCB Communication Mechanisms

The OpenVMS/SEVMS TCB uses a number of mechanisms for its own internal communication. In a uniprocessor configuration, synchronization of access among processes to critical data structures needs to be maintained. This is done primarily by raising and lowering the processor Interrupt Priority Level (IPL). In a multiprocessor configuration, spinlocks are used in addition to the IPL within a node. For code that is not performance critical, a mutex, lock, or other IPC mechanism may be used.

For a VMSCluster system, the TCB uses DEcnet or Systems Communications Architecture (SCA) to coordinate access to cluster-wide resources, including the security databases. These two architectures provide the communication services within a VMSCluster. The DEcnet architecture provides services that are not restricted to a VMSCluster; however, no external connections are permitted in the evaluated configuration. The communication hardware is protected from direct user access by the DAC device protections, as described on page 146, “Device Objects”. The CI and Network Interconnect (NI) controllers are allocated exclusively to system use during the boot process. The system then creates pseudo-devices for use by untrusted processes.

The following sections discuss the use of IPL, spinlocks, mutexes, SCA, and DEcnet. Other IPC mechanisms that are also available to untrusted processes, including the lock manager, are discussed on page 104, “Interprocess Communication”. 

FINAL: 24 October 1996
4.2.1 Interrupts and Spinlocks

The interrupt mechanism of a VAX or Alpha processor provides for both hardware and software interrupts. A software interrupt is recognized as an interrupt by the hardware, but requested by a software instruction rather than a hardware condition. The software interrupts are used throughout OpenVMS/SEVMS to schedule system operations. The process scheduler is the IPL 3 interrupt service routine. This means that three software interrupt levels exist (0 through 2) that can be interrupted by the scheduler. These three levels therefore execute within a given process context. Only IPL 0 and IPL 2 interrupts are currently used. Most user code runs at IPL 0, which is the level in the absence of any interrupt. IPL 2 is used as the underlying hardware support mechanism for Asynchronous System Traps (ASTs). ASTs allow a process to interrupt itself and execute some code, or for the TCB to force a given process to run code in its context.

4.2.1.1 Process Context Interrupts

An AST is a mechanism to signal an asynchronous event to a process and is used for intraprocess synchronization. On completion of the specified event, an AST is delivered to the process. This causes a routine (specified by either the process or the system) to be executed within the context of the process. Typical uses of an AST are as follows:

- To allow a program to continue executing while awaiting an external event
- To allow one part of a program to signal another part of a program and possibly pass data
- To allow a part of the system to communicate with a process to do something in the context of that process.

When an REI instruction is executed, a check is made to determine whether there are pending ASTs to be delivered to the current process. If there are, the REI instruction generates an IPL 2 interrupt so that the AST dispatcher (the IPL 2 interrupt service routine) can deliver the AST to the process. The IPL 2 interrupt service routine then calls the AST routine. The IPL 2 interrupt will not be generated if the access mode of the process is more privileged than the access mode of the AST or if the interrupt stack is active. Multiple ASTs can be queued to the same process and the queue is ordered by access mode (most to least privileged). ASTs can only be interrupted by a second AST when the interrupting AST is more privileged. The ASTIVL register is a per-process hardware register that contains the access mode value for the AST in the queue that has the most privileged access mode. Under some circumstances, a field in the AST Control Block (ACB) is used to queue a second related AST along with the first AST.

The primary data structures involved with the AST mechanism include the ACB and the Process Control Block (PCB). The ACB holds the Process ID (PID) of the target process, a pointer to the AST routine to be executed, and the access mode in which the AST routine is to be executed. The software PCB contains a pointer to the list of ACBs queued to a process. The software PCB also maintains information on active ASTs, the modes for which ASTs are enabled, and the number of ASTs that the process can request.

ASTs are created by three different mechanisms. ASTs may be created by a process requesting notification of completion of an asynchronous system service, such as Queue I/O Request ($QIO). ASTs may be created by the Declare AST ($DCLA$AST) system service. $DCLA$ST allows the calling process to queue an AST for itself at its current access mode or at a less privileged access mode. ASTs may be created by the system queuing an AST to execute code in the context of that process, such as process suspension or process deletion.
4.2.1.2 System Context Interrupts

The interrupt mechanism provides hardware-enforced interprocess synchronization. OpenVMS/SEVMS uses the ability to raise the processor interrupt priority to a given level to synchronize access to certain data structures. A number of specific hardware interrupt levels are associated with data structures. The interrupt level used by a device will also be used to lock access to the device data structures. Other specific levels include “powerfail” IPL (the highest level) and “fork” IPL. After an I/O interrupt is received, the interrupt handler will do only the most time-critical functions. It will then add this I/O to a “fork queue” for a lower level interrupt handler. The original handler will then request a fork IPL interrupt and execute an REI instruction. This process is known as “fork dispatching.”

4.2.1.3 Spinlocks

A processor IPL is maintained for each processor in a multiprocessor node. Therefore, IPL alone is insufficient in these nodes to synchronize access to shared memory locations. A spinlock is simply a test and set bit maintained for a given resource. The routine that coordinates access is one of the routines in the loadable part of the system executive (see page 73, “VMS Executive”). It includes only IPL code if loaded on a uniprocessor, but both IPL and spinlock code if loaded on a multiprocessor. A number of spinlocks mirror common IPL locks (e.g., fork lock to access fork IPL structures).

4.2.2 Mutual Exclusion Semaphore

A mutex (or mutual exclusion semaphore) is a data structure stored in system space. All OpenVMS VAX mutexes and some OpenVMS Alpha mutexes consist of a single longword. OpenVMS Alpha adds support for quadword mutexes which can be locked and unlocked with fewer memory operations than longword mutexes. A mutex is used to control synchronization to data structures accessed by multiple processes. A mutex allows one writer or multiple readers on a data structure. A lock is placed on a mutex before accessing the controlled data structure. The mutex contains information such as the number of processes that have locked the mutex and status flags. A process must stay at IPL 2 or higher, while it has a mutex to prevent an AST from being sent to this process that would cause it to be deleted or placed in an involuntary wait state. If a process must wait because mutex access cannot be granted (the process wants to write and there are processes currently reading), the process is put in a wait state. There is no direct user interface to a mutex. Mutexes are used where synchronization is required at low IPLs. One example is access to the shared logical name data structures, where access at or above IPL 3 would prevent faulting in non-resident pages.

4.2.3 Interprocess Communication System Service

The $IPC system service provides a consistent interface for the TCB to use various forms of inter-process communication within a VMScluster. It requires SYSNAM privilege to use and is therefore accessible only to the TCB. This is an interface currently used only by the queue management server (see page 127, “Queue Management”). The underlying mechanisms are either DECnet services, SCA services, or a local buffer (for strictly intra-node use).
4.2.4 System Communication Architecture

SCA is a framework for communications within a VMScIuster system with functions that are tailored to the internode communication needs of a VMScIuster system. SCA defines a set of communication services for use across the cluster. These services are known as System Communication Services (SCS). SCS communicates with system applications (SYSAPs) on its local VAX node, and formats requests to other node's SYSAPs via local port drivers. The basic structure of SCA is shown in Figure 4.1. Only the software for a single node is shown in the figure. For VAX, SCsLOA is a VMS executive image that contains SCS proper. For Alpha, the name of this image is SYS$SCS.

4.2.4.1 Port Drivers

SCS operates over several varieties of communication hardware. Specifically, these are high speed CI hardware, standard Ethernet, Digital Storage Systems Interconnect (DSSI), and Fiber Distributed Data Interface (FDDI). OpenVMS/SEVMS includes a specialized driver for the CI and DSSI hardware. The Ethernet and FDDI hardware is accessed via a specialized driver that utilizes the services of more generic network drivers.

4.2.4.2 SCS Applications

An SCS application is a function within the TCB of OpenVMS/SEVMS that communicates via SCS.

The Connection Manager provides coordination for the VMScIuster nodes. Collectively, the connection manager system applications regulate membership of the cluster. Each connection manager decides if the cluster is appropriately formed from its local perspective and informs the others. The connection manager, for example, requires that it has a direct connection to every other connection manager, and that a minimum number of nodes be available to boot. These requirements ensure that the VMScIuster has not split into disjoint pieces that each have access to shared resources such as disk drives. If the connection manager cannot ensure that the requirements have been met, it will stop the boot process of the node and wait for conditions to change while printing messages to the node's console.

The System Management Server provides an interface for operators and administrators to perform functions on multiple systems simultaneously. In SEVMS, these functions are restricted to operate only at system low classification.

The Lock Manager is used to coordinate resource locking across the entire cluster. For more information on the lock manager system, see page 106, “Resource Locks”.

The Cluster Server provides a process on each VAX node that can be utilized by the other nodes for certain distributed commands (mount, dismount, broadcast, set, and process management functions). It is started only on VMScIuster nodes during system startup, and runs with all privileges.
Figure 4.1. System Communication Architecture Block Diagram for VAX.
The MSCP Servers provide a block I/O interface to disks and tapes. The MSCP server software in a VAX node provides this interface to other nodes for controllers on a node’s bus that do not directly support MSCP. The MSCP server software performs local I/O requests for disk class driver SCS applications. This interface is provided by the MSCP server for I/O controllers in the UDA, KDA and KDB series. It is also provided by HSC software managing disk and tape I/O and DSSI controllers.

The TMSCP Servers provide a slightly modified MSCP service that is tailored to tape operation.

The Disk Class Drivers translate requests from a node’s I/O subsystem for disk blocks into MSCP commands to be performed by an MSCP server, HSC, or DSSI controller.

The Tape Class Drivers are the magnetic tape equivalent to the disk class drivers.

4.2.5 DECnet

The function of DECnet is to communicate between processes on different nodes. It is designed as a general network protocol; in the evaluated configuration, its use is restricted to a VMScluster system. The OpenVMS/SEVMS TCB makes use of DECnet to perform remote booting of satellite systems and to provide system management services across multiple nodes. DECnet TCB interfaces can be used to deliver mail and phone messages to remote nodes (see page 109, “MAIL and PHONE Utilities”), to access remote files (see page 118, “Accessing Files In A Cluster”), to perform task-to-task communication between processes on different nodes (see page 109, “DECnet”), and to login to a remote node (see page 173, “Login Processing”).

DECnet provides connection-oriented communication from a process on one node to an application process on another node. When a connection is requested, DECnet enforces controls on both the local and remote nodes. The controls are based on parameters established by system administrators and stored in the OpenVMS/SEVMS security database. In the evaluated configuration, all nodes share a single set of DECnet parameters. In addition, SEVMS restricts connections to the same MAC classification unless the DECnet application possesses both the BYPASS and DOWNGRADE privileges.

4.2.5.1 Network ACP

The Network ACP (see page 126, “Ancillary Control Processes”) provides the communication services between network hardware devices and processes that wish to communicate. A process requesting a DECnet connection issues an I/O request to its local Network ACP. The local Network ACP verifies that the process is authorized for an outgoing connection to the specified DECnet application and then transmits the request to the Network ACP on the remote node. The remote Network ACP associates an OpenVMS/SEVMS user with the connection request and creates a process to execute the requested DECnet application (or, in some cases, locates an existing process to handle the request). The username and account characteristics associated with the created process are determined based on set of mechanisms described on page 174, “Authentication for DECnet Applications”.

85

FINAL: 24 October 1996
CHAPTER 4  SOFTWARE ARCHITECTURE

4.2.5.2 DECnet Applications

A DECnet application is a set of attributes for a network connection. The attributes include the name of a process or executable image to perform the application processing. OpenVMS/SEVMS privileges may also be designated as a requirement for using a DECnet application. The applications and their attributes are defined by a system administrator using the Network Control Program (NCP) utility. This section provides an overview of the DECnet applications provided by OpenVMS/SEVMS.

The File Access Application supports access to remote files. The File Access Listener (FAL) program accesses files on its node (using RMS) in response to a request from a remote node. Access control to the files is provided on the node where the file resides. This is different than the method used by MSCP connections which rely on access control at the requesting node.

The Mail Application processes connections for the purpose of mail transmission.

The Phone Application processes connections to a real-time user communication utility (the PHONE utility).

The Task Application supports connections to execute a Digital Command Language (DCL) command procedure. Once the connection is established, task-to-task communication between the local and remote processes can be performed using standard OpenVMS/SEVMS I/O operations.

The Remote Terminal Application accepts connections from other nodes and sets up a pseudo-terminal for the incoming connection. It then passes control to a device driver to handle the connection as a login. When the device driver attempts to “hang-up” at the end of the session, the remote terminal application is invoked again to terminate the connection.

The Network Management Application processes connections to the network manager for a VAX or Alpha node.

The Performance Monitor Application supports connections to a process that is collecting performance statistics on a VAX or Alpha node.

4.2.5.3 Maintenance Operation Protocol

A DECnet communication protocol, Maintenance Operations Protocol (MOP), is used to boot diskless nodes, primarily workstations. The boot process can be initiated by the Network Management Application or by an unsolicited network request. For the unsolicited case, a booting workstation sends a request to all nodes on the communications media. Each node receives the message through its network ACP and creates a maintenance process. If the node receiving the request has an entry in its database for the satellite’s network address, then it will respond to the satellite. The first node to respond to the request is used for all
4.3 Memory Management

The two functions of memory management in OpenVMS/SEVMS are to manage the memory within an address space and to provide each process with memory for the data and programs. Each process is provided the illusion that it has a physically contiguous 4 Gbyte address space. The primary unit of memory managed by OpenVMS/SEVMS is a page. For VAX, virtual memory is divided into pages of 512 bytes, and physical memory is also divided into 512-byte page frames. For Alpha, allowable page sizes can be 8K bytes; 16K bytes; 32K bytes; or 64K bytes. The default is 8K bytes. The same granularity is applied to primary memory. Pages of the process’ address space may exist in physical memory in a node, on disk, or, in the case of zero-filled pages, may not exist in physical memory at all.

Each process has a “working set” of pages. This set of pages is in physical memory and allocated to the process. The pages of a process’s virtual memory that are not in its working set are accessible either as real memory in the system that has not been flushed to disk, or in a disk file. The pages may be part of the process’s image (executing) file, or part of the system page or swap files. The page file contains small collections of pages of process data, while the swap file contains whole (or nearly whole) processes that have been removed from the system’s real memory. The number of pages in a process’s working set may shrink and grow as the system responds to the demand (or lack of demand) for memory by the all the processes on the system.

For VAX, P0 and P1 (the lower half of the address range, 0 - 7ffe ffff16) are called per-process space, and S0 and S1 are called system space. For Alpha, P0 occupies the address range 00000000 00000000 - 00000000 3fffffff16, P1 occupies the address range 00000000 40000000 - 00000000 7fffffff16, and the System Space occupies the address range ffffffff 80000000 - ffffffff ffffffff16.

Per-process address spaces are created for each process while system space is created only once and mapped into every process’s virtual address space. Separation between per-process address spaces is by means of the per-process page tables. The virtual address space is composed of pages. Pages are grouped into sections. Sections are the unit of memory usually handled by memory management, and are combined to form images. The following sections will discuss mapping virtual addresses to physical addresses, pages, sections, images, and the management of the virtual address space.

4.3.1 Pages

Each page in physical memory is located by its physical address (the Page Frame Number (PFN)). The PFN database contains several arrays of data. Each array contains an entry for each page in physical memory that is subject to paging. In two cases, arrays are used for dual purposes to save space. These arrays contain forward and backward links to other pages if the page is not mapped. If the page is mapped, these arrays are used to store other information. Pages used by the executive that are not subject to paging are not mapped in the PFN database. The arrays are indexed by the PFN. The following list describes the information kept in each array:
CHAPTER 4. SOFTWARE ARCHITECTURE

- TYPE array (VAX only): contains the type of virtual page that occupies this physical page (i.e., process page, system page, or page table page). For Alpha, this is folded into the PAGE.STATE array.
- PTE array: contains the virtual address of the page table entry that maps to the page.
- BAK array: contains information about the backing store for the virtual page occupying the page frame. The backing store is the location of the page when it is not memory-resident (i.e., the page file or the original image file).
- PAGE.STATE array: contains information used to help locate the page, such as whether the page frame is on the free or modified page list, whether the page is active, whether a write to backing store is in progress, or whether a read from backing store is in progress.
- FLINK and SHRCNT array: when the page is not mapped by a virtual page the array contains a forward link to the modified, free, or bad page lists; when the page is mapped by a virtual page the array contains the number of process PTEs that map to a global page.
- BLINK and WSLX array: when the page is not mapped by a virtual page the array contains a backward link to the modified, free, or bad page lists; when the page is mapped by a virtual page the array contains a working set list index for the page.
- REFCnt array: contains the number of reasons a page shouldn’t be placed on the free or modified page lists. For example, the count is incremented for every process working set that contains the page and every incomplete I/O involving the page.
- SWPVBn array: contains a virtual block number in the swap file where a locked page goes when the process has been swapped out and there is I/O in progress.

The per-process page tables are contained in a process’ Process Header (PHD) in system space. These tables contain the complete description of the per-process virtual address space and are both pageable and swappable. Where the PFN database organized pages by physical location, the page tables organize pages into virtual address spaces regardless of their current physical location.

4.3.1.1 The Process Page Tables

The process page tables contain Page Table Entries (PTEs) for all pages currently defined in process virtual space. The P0 page table contains Page Table Entries (PTEs) for the pages in P0 space; the P1 page table contains PTEs for the pages in P1 space. A PTE for a valid page contains the PFN (the page’s physical address) of the page. A valid PTE for VAX is shown in Figure 3.4.

An invalid PTE can contain a Process Section Table Index (PSTX), a page file virtual block number, a Global Page Table Index (GPTX), a PFN, or an indication that it is a demand zero page. A PTE with a PSTX has the index of a Process Section Table Entry (PSTE). A PTE with a page file virtual block number points to the block in one of four page files containing the referenced page. A PTE with a GPTX contains an index into the Global Page Table (GPT) where an associated global PTE is used to locate the page. If a page is in transition (removed from the working set and placed on the free or modified lists, or in transit between mass storage and physical memory) the PTE contains a PFN. A PTE with a 0 for the PFN indicates a demand zero page that is filled with zeroes on a page fault. Figure 4.2 shows an example of an invalid page table entry for VAX used by the memory management software, but that cannot be used directly by the hardware.

The System Page Table (SPT) contains System Page Table Entries (SPTEs) that describe 50 virtual address space. The SPT is located in non-pageable memory. The SPT is the basis for any virtual address translation because per-process page tables are located in system virtual space. The valid and invalid (SPTEs) are the
same as for PTEs, except that an invalid System Page Table Entry (SPTE) cannot have a GPTX.

4.3.2 Sections

As previously mentioned, a PTE can contain a PSTX, a page file virtual block number, a GPTX, a PFN, or an indication that it is a demand zero page. In the case of a PFN, information on where the page will be stored if required to be paged out, is stored in the PFN database. This information will be supplemented and the PTE updated if the page is actually paged out. A PTE that is demand zero has never been written. If it is written to, it will have a PFN or a page file virtual block number. PTEs that contain either a PSTX or GPTX index into a table of entries describing sections. A section is a collection of pages in an image file. A data file may also be mapped as a section. Sections are mapped by memory management into pieces of virtual address space as contiguous pages. Process sections and global sections are discussed below.

4.3.2.1 Process Sections

The Process Section Table (PST) is contained in the PHD and contains PSTEs. A PSTE describes the association between a portion of virtual address space and a portion of an image. A section may be from an image file, a sharable image file, a process's data file, or a page file.

4.3.2.2 Global Sections

Global sections provide a mechanism for multiple processes to map a data area into each of their own address spaces. A global section can be a disk file (or portion of a disk file), a set of physical addresses represented by page frame numbers (a PFN-mapped section), or a set of demand zero pages whose pages are backed in a page file or disk file. Global sections may be mapped and shared by multiple processes.

Global sections are mapped into P0 or P1 space. The mapping may be different for each process using a section. The data structures describing the global sections are maintained in system space. When a global section is created, a Global Section Descriptor (GSD) is allocated and loaded with information such as the global section name, the User Identification Code (UIC) of the section creator, the UIC of the file owner and access control information. A Global Section Table Entry (GSTE) and Global Page Table Entry (GPTE) are built for memory management purposes. The GSTE serves the same purpose as the PSTE and the GPTE serves the same purpose as the PTE. The GSD associates the global section name to the GSTE. The GSD is only used when a process attempts mapping to or destruction of a global section or when the security characteristics are read or modified. When a global PFN-mapped section is created, only the global section
CHAPTER 4. SOFTWARE ARCHITECTURE

descriptor need be created.

For a global section, a process’s PTE points to a GPTE in the global page table. The translation from
PTE to GPTE is performed in OpenVMS/SEVMS software rather than in the hardware. When there is a
page fault on a global page, the pager locates the GPTE using the GPTX stored in the process’s PTE and
then proceeds as described above to read the page into memory. If the page is already in memory (because
another process is already accessing the page) the pager simply validates the PTE from the GPTE data and
updates the count of processes accessing the global page.

The Create and Map Section (§CRMPSC) system service creates either a private section or a global section
and optionally maps it. Global sections must be supplied with a character string name so that other processes
may identify the global section to be mapped. Sections may be designated read-only or read-write.

The Map Global Section (§MGBLSC) system service maps an existing global section to the current process.
Each of the designated process PTEs is marked invalid and loaded with a global page table index. If the
section isPFN-mapped, each process PTE is marked valid and the PFN field is loaded with the physical
page number. The Delete Global Section (§DGBLSC) system service marks an existing global section for
deletion. The actual deletion occurs when all processes that had mapped the global section have deleted the
mapped pages.

There are system and group global sections and these can be temporary or permanent. A temporary global
section does not require any privilege to create, and is deleted when there are no more processes mapped
to it. The PRMGBL privilege is required to create or delete a permanent global section. No privileges are
required to create a group global section, while the SYSGBL privilege is required to create or delete a system
global section. The main difference between system and group global sections is that group global sections
are named relative to the creator’s Group Identifier (GID). There is no interface to group global sections
outside a process’s own group. System global sections may be referenced by any process. Both types of
global sections have DAC protection, and, in SEVMS, MAC protection. PFNMAP privilege is required to
create or delete a PFN mapped section.

4.3.3 Images

Images are made up of an image header (IHD for VAX; EIHD for Alpha), image sections (program code and
data), a fixup table (information for address references which must be resolved at activation), and possibly
symbol table information. The sections are combined into the VMS executive or mapped into a process’s
address space. Each section has a set of attributes in the process header including the page protection
codes for the pages of the section. The attributes also specify if the section contains executable code or if it
contains data. The image header may (and often does) include a description of one or more sections that do
not exist in the image, but that should be allocated as zero filled pages when the image is invoked. Process
management utilizes images to fill the process address space with executable code, as described on page 99,
“Images”.

4.3.4 Virtual Address Mapping

When a process refers to a memory location in its virtual address space, the information being referenced
may be in memory, on disk, or not yet allocated. When a reference is made to a memory location, the
hardware performs an access check and then translates virtual addresses into physical addresses. Processes
have sets of page frames called a Working Set List (WSL). The WSL contains those pages of the process’s virtual memory that are currently valid for hardware translation to physical memory.

For invalid PTEs, the page is either all zeroes, or part of a disk file. The disk files that contain pages may be image files or data files, as described under “Sections” above, or, in the case of modified pages, are likely in the page or swap files. The page and swap files are maintained by the pager and swapper and are owned by the system. Only processes in the system group or with privilege can access the files.

When the PTE for the referenced page is invalid, the pager is activated as a result of the hardware fault. It is the pager’s function to attempt to have the pages in memory that are most likely to be referenced by swapped-in processes. The pager will move pages from a process’s working set to make room for pages that are being paged in. The swapper is signaled when it becomes necessary to move pages no longer in working sets out of physical memory to make room. The swapper may decide to make room by moving entire processes out of memory. It is the function of the swapper to attempt to have the most likely processes to be scheduled for execution in memory.

4.3.4.1 Swapper

The swapper’s main functions are to write modified pages to page files, to shrink working sets when necessary, and to swap process in and out of memory. The swapper is a process whose code is part of an executive loadable image. The swapper executes in the kernel mode of the processor and resides in S0 space.

The swapper maintains a minimum number of free pages for the pager to use by writing modified pages to page files, reclaiming pages from deleted process headers, trimming working sets, and by outswapping processes. The reasons the swapper is invoked include an outswapped process becoming computable, a process reaching a quantum end, a process reaching its total Central Processing Unit (CPU) time expiration, the modified page list becoming too large, the free page list becoming too small, or a timer signaling that the swapper be invoked.

The swapper balances demands for physical memory from processes and pageable portions of the operating system. Needed pages are always allocated from the free page list. Swaps are transfers of entire working sets between memory and disk. When a process is swapped out, the swapper puts available pages in the free page list. After a swapout, the software process control block and job information block are the only process information retained in physical memory. Reasons for swapping out are to reclaim free space and to free up balance slots.

Processes are moved to a swap file in two parts, first the process body (P0 and P1 pages) and then the PHD. There may be partial swaps where the process header stays in memory because there are locked pages for the process (e.g., direct I/O is in progress). To swap a process in, the PHD is brought into memory first and then the process body.

4.3.4.2 Pager

The pager’s main function is to bring pages from disk into physical memory. The pager is an exception service routine executed in the context of the process that caused the page fault. When an invalid page is referenced by a process (a page fault), an exception is raised and the pager executes. The pager retrieves the referenced address from the kernel stack (placed there by the hardware when the page fault occurred),
and determines the page’s location using the page tables. The pager makes use of the process section table, the page tables, the working set list, and the system page table when performing its functions.

When a page is no longer needed by a process or is taken from a working set, the page may go to one of two places. If the page was not updated, it is placed in the free page list; if the page was updated, it is placed in the modified page list. When the free list is too small, modified pages are written to a paging file by the swapper, and then placed on the free list. Pages are placed on these lists rather than immediately reused to alleviate paging I/O (on a page fault a referenced page may still be in memory in one of the page lists).

On a page fault, process private pages can be located in a file (the PTE contains a PSTX), can still be in physical memory (on the free or modified page lists), can be a demand zero page (process is requesting an empty page) or can be located in a page file (the page was paged out). When the page is in a file, the pager locates the page using the PSTX, creates and queues an I/O request, allocates a page from the free list and makes room for it in the processes working set, waits for the I/O to complete, and updates the PFN database. For a page still within physical memory (e.g., the page is in the free page list), the pager changes the state of the page to active, marks the PTE valid, and returns. For demand zero pages, the pager allocate a page from the free page list, initializes the PFN database, fills the page with zeroes, marks the PTE valid, and returns. For pages located on a page file, the pager performs the same functions as described above for a page in a file.

### 4.3.5 Management Within a Virtual Address Space

The structure of the VAX virtual address space was described earlier (see Figure 3.2, page 27). OpenVMS/SE-VMS uses each of the virtual address regions for different purposes. P0 space (shown in Figure 4.5) contains the user’s code and data (the process image) and is managed by each process. P1 space (shown in Figure 4.4) contains process control information maintained and protected by the TCB, including the process stacks and system service vectors. S0 space (shown in Figure 4.3) contains operating system code and data; S1 space is reserved and unused. On the 6000 and 9000 series VAX CPUs, the hardware length register for S0 space is extended by one bit. This means that the S0 virtual address space can be twice as large on these processors. Consequently, the virtual addresses reserved for S1 are part of S0 space on these processors. OpenVMS/SEVMS utilizes the additional space as it would S0 space (e.g., larger pools and more process headers).

The structure of Alpha virtual address space was described earlier (see Figure 3.9, page 44). “System Space” (shown in Figure 4.6) contains operating system code and data. P1 space, which contains process control information maintained and protected by the TCB, including the process stacks and system service vectors, is shown in Figure 4.7.

OpenVMS/SEVMS maintains areas in system space (S0 for VAX; “System Stack” for Alpha) called pools, from which it allocates and deallocates data structures. System space has non-paged and paged pool areas. Non-paged pool contains variable length data structures such as interrupt service routines, device drivers, and process control blocks. Non-paged pool also contains fixed-length lookaside lists to hold data structures such as request packets for sending and receiving messages. Paged pool contains data structures that are used by multiple processes, but that are not required to be permanently memory-resident. Both pools can be read from executive and kernel mode but written only from kernel mode.
### 4.3. MEMORY MANAGEMENT

<table>
<thead>
<tr>
<th>System Service Vectors</th>
<th>SYS.EXE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executive Transfer Vectors</td>
<td></td>
</tr>
<tr>
<td>System Data Area</td>
<td></td>
</tr>
<tr>
<td>SYSGEN Parameters Area</td>
<td></td>
</tr>
<tr>
<td>Boot Parameters Area</td>
<td></td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
</tr>
<tr>
<td>Images Loaded by SYSINIT</td>
<td></td>
</tr>
<tr>
<td>Images Loaded by EXEC_INIT</td>
<td></td>
</tr>
<tr>
<td>Images Loaded by SYSBOOT</td>
<td></td>
</tr>
<tr>
<td>More Images Loaded by SYSINIT</td>
<td></td>
</tr>
<tr>
<td>I/O Pages</td>
<td>Areas not mapped in PFN database (non-paged)</td>
</tr>
<tr>
<td>Pageable System Pages</td>
<td></td>
</tr>
</tbody>
</table>

**Restart Parameter Block**

| PFN Database |         |
| Paged Pool |         |
| Nonpaged Pool |         |
| Per-CPU Databases |         |
| Unmapped Page |         |
| Boot Stack |         |
| Unmapped Page |         |
| Interrupt Stack |         |
| Unmapped Page |         |
| System Control Block |         |

**Balance Slots**

| System Header |         |
| System Page Table |         |
| Global Page Table |         |
| SMP Boot P0 Page Tables |         |

Figure 4.3. VAX S0 Space
### VAX P1 Space

<table>
<thead>
<tr>
<th>Static portion</th>
<th>Dynamic permanent portion</th>
<th>Image-specific portion</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Stack (grows toward low addresses)</td>
<td>Per-process Message Section</td>
<td>CLI Symbol Table</td>
</tr>
<tr>
<td></td>
<td>CLI Common Table</td>
<td>CLI Image</td>
</tr>
<tr>
<td></td>
<td>Files-11 Data</td>
<td>Files -11 Image</td>
</tr>
<tr>
<td></td>
<td>Image I/O segment</td>
<td>Process I/O segment</td>
</tr>
<tr>
<td></td>
<td>Process Allocation Region</td>
<td>Channel Control Block Table</td>
</tr>
<tr>
<td></td>
<td>P1 Window to Process Header</td>
<td>RMS Data Pages</td>
</tr>
<tr>
<td></td>
<td>Per-process Common Area for Users</td>
<td>Per-process Common Area (reserved to Digital)</td>
</tr>
<tr>
<td></td>
<td>VMS User Mode Data Page</td>
<td>Security Audit Data Pages</td>
</tr>
<tr>
<td></td>
<td>Generic CLI Data Pages</td>
<td>Generic CLI Data Pages</td>
</tr>
<tr>
<td></td>
<td>Image Header Buffer</td>
<td>Image Header Buffer</td>
</tr>
<tr>
<td></td>
<td>KRP Lookaside List</td>
<td>KRP Lookaside List</td>
</tr>
<tr>
<td></td>
<td>Kernel Stack</td>
<td>Kernel Stack</td>
</tr>
<tr>
<td></td>
<td>Executive Stack</td>
<td>Executive Stack</td>
</tr>
<tr>
<td></td>
<td>Supervisor Stack</td>
<td>Supervisor Stack</td>
</tr>
<tr>
<td></td>
<td>System Service Vectors</td>
<td>System Service Vectors</td>
</tr>
<tr>
<td></td>
<td>Debugger Symbol Table</td>
<td>Debugger Symbol Table</td>
</tr>
</tbody>
</table>

**Figure 4.4. VAX P1 Space**

*Final Evaluation Report Digital Open VMS and SEVMS, Version 6.1, with VAX or Alpha*
<table>
<thead>
<tr>
<th>Unmapped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable Image</td>
</tr>
<tr>
<td>LIBRTL</td>
</tr>
<tr>
<td>LBRSHR</td>
</tr>
<tr>
<td>Other Sharable Images</td>
</tr>
<tr>
<td>Debugger (if requested)</td>
</tr>
<tr>
<td>Traceback (if requested)</td>
</tr>
<tr>
<td>Unmapped</td>
</tr>
</tbody>
</table>

Defined by linker mapped by image activator.

Not defined at link time.
Both cannot be mapped at the same time.

Figure 4.5. VAX P0 Space
<table>
<thead>
<tr>
<th>Nonpageable Code Sections from Executive Images and Images Installed Resident</th>
<th>FFFFFFFF 80000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonpageable Data Sections from Executive Images</td>
<td></td>
</tr>
<tr>
<td>System Header</td>
<td></td>
</tr>
<tr>
<td>PFN Database</td>
<td></td>
</tr>
<tr>
<td>Error Log Allocation Buffers</td>
<td></td>
</tr>
<tr>
<td>Nonpaged Pool</td>
<td></td>
</tr>
<tr>
<td>Nonpaged Pool Expansion Area</td>
<td></td>
</tr>
<tr>
<td>Balance Set Slots</td>
<td></td>
</tr>
<tr>
<td>Global Page Table</td>
<td></td>
</tr>
<tr>
<td>Paged Pool</td>
<td></td>
</tr>
<tr>
<td>System Control Block</td>
<td></td>
</tr>
<tr>
<td>Hardware Restart Parameter Block</td>
<td></td>
</tr>
<tr>
<td>Console Routines</td>
<td></td>
</tr>
<tr>
<td>Boot Driver Data Areas</td>
<td></td>
</tr>
<tr>
<td>Physical Memory Bitmaps</td>
<td></td>
</tr>
<tr>
<td>Guard Page</td>
<td></td>
</tr>
<tr>
<td>Primary Processor’s System Context Kernel Stack</td>
<td></td>
</tr>
<tr>
<td>Guard Pages</td>
<td></td>
</tr>
<tr>
<td>Primary Processor’s Machine Check Logout Area</td>
<td></td>
</tr>
<tr>
<td>Guard Page</td>
<td></td>
</tr>
<tr>
<td>Available System Pages</td>
<td></td>
</tr>
<tr>
<td>System Space Expansion Region</td>
<td></td>
</tr>
<tr>
<td>System Page Table</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.6. Alpha System Space
4.3. MEMORY MANAGEMENT

Figure 4.7. Alpha P1 Space
4.4 Process Management

A process in OpenVMS/SEVMS is the entity that is scheduled for execution and is fully described by data structures specifying the hardware context, software context, and virtual address space. A process can be either detached or a subprocess of the process that created it. A job is a detached process and all of its descendant are subprocesses. This section describes process creation, the process data structures, images and image creation, process deletion, process scheduling, interactive and batch process creation and destruction, and process privileges.

4.4.1 Process Data Structures

OpenVMS/SEVMS maintains a number of data structures to describe processes and jobs. For each job there is a Job Information Block (JIB) stored in system space (S0 space for VAX: System Space for Alpha), virtual address space that contains a username and a set of process quotas that are shared among all processes in the job. An example of such a quota is the number of bytes buf awaiting I/O.

Each process has an associated software PCB, hardware PCB, and the PHD. The software PCB is located in non-paged pool and is memory resident for the life of the process. It contains information such as the PID, the User Identifier (UID) of the owner, the address of the PHD, the address of the JIB, the address of the Access Rights Block (ARB) (see page 135, “Subjects”), the address of the hardware PCB, and the current process state (e.g., computable, current).

The hardware PCB contains four stack pointers (one for each access mode), process registers, the program counter, the program status longword, and the P0 and P1 base and length registers. All the current process information above except for the memory management registers is copied to the hardware PCB fields during a context switch with the Save Process Context (SVPCTX) instruction (VAX). The Alpha instruction is SWPCTX. When there is a change to the memory management registers, the system updates both the registers and the hardware PCB fields.

The vector processor for VAX systems supporting this feature has a much larger hardware context than a scalar VAX processor. The vector context is saved and/or restored only when a different process executes a vector instruction.

The PHD contains the P0 and P1 page tables, a pointer to the software PCB and the working set list. The PHD is swappable, is located in system space, and has both pageable and non-pageable components.

Balance slots are sections of system virtual address space reserved for a PHD. A system generation parameter defines the number of real balance slots. The number of the balance slot in which a process’s PHD resides is kept in the fixed (non-paged) portion of the PHD. An executing process must be in a real balance slot. However, for memory-resident processes that are not executing, the swapper maintains some process’s PHDs in system virtual (S0) address space, but not in a real balance slot. These PHDs are restored to real balance slots whenever the process is required to execute.

The Process Quota Block (PQB) is a temporary data structure used only during process creation. It holds information needed during the creation since there is not an address space yet. The PQB contains such information as privilege masks, quotas, authorization information, and Command Language Interpreter (CLI) name (see page 102, “CLI Processing”).
4.4.2 Process Creation

All processes (except the SWAPPER) are created by an existing process via the Create Process ($CREPRC) system service. The $CREPRC system service performs the following steps:

1. Makes privilege checks and quota checks.
2. Allocates a PCB, JIB (if creating a detached process), and a PQB.
3. Initializes the PCB, JIB, and PQB using explicit $CREPRC arguments and implicit parameters taken from the context of the caller.
4. Assigns the UIC and security attributes to the process (see page 135, “Subjects”).
5. Places the new process into the scheduler database.

The swapper builds a basic process from a process shell in the VMS executive, builds the PHD, and requests the new process be scheduled for execution. The shell contains a PHD, P1 page tables, P1 pointer page, an RMS data area, and initialization code. The routine EXE$PROCSTART is invoked, and it copies information from the PQB and PHD into P1 space, calls the image activate system service to activate an image in the process, changes to user access mode, and calls the image at its entry point. From this point on, the only way a process can change access modes is by activating privileged images that have the authority to change modes, having an AST delivered in a more privileged mode, or by issuing the CHMK or CHME instructions that transfer control to system services.

The system service $CREPRC uses information passed by the caller and also copies information from the caller’s context. Information copied from the caller includes the caller’s UIC, privilege masks, and authorization information.

There are three instances in OpenVMS/SEVMS outside of administratively privileged processes where $CREPRC is called by the TCB. These are the job controller process, the NETACP process, and the DCL interpreter. The job controller creates processes in response to external events at terminals and card readers to authenticate users. The NETACP process creates processes to run DECnet applications and is discussed on page 85, “Network ACP”. The DCL interpreter creates processes at the request of the user without using any privileges not possessed by the user.

4.4.3 Images

An image is a collection of code and data sections as described on page 90, “Images”. An image may be a DCL command, a privileged TCB program, or a user program. A main image is what is typically thought of as a program. It cannot be linked with other images and it may contain one or more sharable images. Sharable images may be shared among processes in memory and can only be linked with main images, not independently executed. This section will describe known images, image activation, image startup, and image rundown (or termination).
4.4.3.1 Known Images

The install utility creates a known image database (or known file database) with a Known File Entry (KFE) for each known image. The entries contain information used by the image activator to locate and map the image. Reasons for having known images include allowing faster image activation to increase performance, identifying images that are shared, and allowing untrusted users access to images that require privilege to run.

An image that needs privileges but that must execute in a nonprivileged process's context is installed as a known image with privilege. When the image is activated, the process gains enhanced privileges temporarily, and the privileges are removed when the image is rundown. The image's privilege mask is stored in the KFE. The install utility also installs images as protected. This allows an appropriately linked image to function as a system service and execute in the protected modes of the CPU.

At system startup, all default known images are installed. In the case of OpenVMS, all known images are installed from records in the VMSIMAGES.DAT file. In the case of SEVMS, some additional images are installed by SEVMS command procedures run during system startup. An administrator can install known images after system startup with the install utility. In the evaluated configuration, only the default images may be installed as protected or with security relevant privileges. The TFM provides appropriate administrative guidance.

An administrator may add protected subsystems (see page 158, “Subsystem ACE”) within the evaluated configuration. This allows an image to maintain and export its own policy with respect to objects owned and controlled by the subsystem. The subsystem maintains DAC control over only the objects owned by the subsystem, and, in SEVMS, cannot bypass MAC. Subsystem identifiers are handled as image privileges by the image activator, allowing their use only while the image is executing.

4.4.3.2 Image Activation

Image activation includes locating the correct image file on disk; setting up process page tables; resolving address references among sharable images; and setting up hooks for debuggers, image dump utilities, or traceback handlers. The OpenVMS/SEVMS internal system services Image Activator ($IMGACT) and Image Address Fixup ($IMGFIX) are used for image activation.

$IMGACT calls RMS to open image files (file protection checks take place at this time), reads the Image Header (IHD) (which has the virtual address space requirements for the image sections), and requests the memory management system service to map image sections into the calling user’s address space. Activation of a main image that has been installed with privilege, or that has subsystem identifiers, extends the process privileges or rights identifier list with the image privileges and identifiers for the life of the image (see page 141, “Privilege Propagation”). Activation of an image usually results in activation of one or more sharable images (i.e., run-time libraries). Sharable images may be installed as protected. A protected sharable image includes the change mode instruction used to invoke it. This instruction is copied into the process’s P1 space system service vector table by the $IMGFIX service.

The $IMGFIX service is responsible for resolving all the references in the image sections by use of the image fixup information added to the image by the linker. The $IMGFIX routine also adds protected sharable images to the processes system service vector list and sets page protections on the pages of an image.
4.4.3.3 Image Startup

After image activation, the image is called at its transfer address. This can be a debugger, user-supplied initialization procedure, or the executable image itself. If an image is linked with the /NOTRACEBACK qualifier, the transfer will always be to the image itself. The install utility ensures that all privileged and protected images are linked with the /NOTRACEBACK qualifier. In addition, images to which a process has only execute (i.e., not read) access cannot have image transfer addresses other than the image.

4.4.3.4 Image Rundown

Image rundown removes all traces of the current image so that a new image may start executing. This allows DCL to have multiple images that execute one after another. The internal OpenVMS/SEVMS system service Rundown Image ($RUNDOWN) is used for this purpose. $RUNDOWN ensures later images do not inherit elevated privileges or get reduced working sets from the previous image. $RUNDOWN is also used for cleanup during process deletion. The actions $RUNDOWN performs are indicated by the calling process’s access mode. If called from user mode, $RUNDOWN removes the current image so that it does not affect subsequent images. This includes resetting image pages in memory (deleting all of P0 space and deleting nonpermanent P1 space pages), deassigning channels, releasing locks, resetting the working set list, and resetting privilege masks. When called from kernel mode, process cleanup is done.

4.4.4 Process Deletion

Process deletion occurs when the Delete Process ($DELPRC) system service is invoked. This service runs in kernel mode with initial operations occurring in the context of the caller (such as privilege checks). Final operations occur in the context of the process being deleted. $DELPRC makes sure file operations are complete, causes image rundown, deletes subprocesses, calls $RUNDOWN to rundown the process, deletes the address space, updates quotas of the owner process and sends a message to the owner process (if the process being deleted is a subprocess), deletes any remaining P1 space, and deletes the process data structures.

4.4.5 Process Scheduling

The PCB contains the scheduling state, process priority, requirements for CPU capabilities, and affinity requirements of a process. All of these process attributes affect process scheduling. Essentially, the highest priority computable process is picked to run, and, when there are two or more computable processes at the same priority, a round robin algorithm is used. CPU capability or affinity requirements are further restrictions on which process is chosen to run and may limit the process' execution to a subset of CPUs in a multiprocessor environment.

For VAX, are 32 priorities for processes on OpenVMS/SEVMS. Priorities 0-15 are for normal processes including all untrusted processes. Normal processes run until they are preempted, enter a wait state, or reach the end of their time quantum. Priorities 16-31 are for real-time processes. A real-time process is not susceptible to quantum end and its priority is not changed dynamically by OpenVMS/SEVMS. The initial priority for a user process is taken from the authorization database and the default is 4. OpenVMS/SEVMS changes current priority over time to favor I/O bound and terminal I/O processes. The privilege ALT PRI
allows changes to priorities; without the privilege a process can only change its priority between 0 and the base priority assigned at process creation. Alpha supports 64 priorities for processes.

The scheduler has a database that includes PCBs and process state queues. The queues are a set of PCBs with the same state and priority. For VAX there are 32 computable (COM) queues, one for each priority, and 32 computable but swapped-out (COMO) queues, one for each priority. There are 64 such queues for Alpha.

Reasons to change the process currently executing include: the process goes into a wait state, the process reaches the end of quantum, the process changes its priority resulting in another process being at a higher priority, the required CPU capabilities no longer match, or the system changes another process to the computable state, and its priority preempts the current process’s priority. When the scheduler initiates a process change, a context switch takes place (see page 22, “Context Switching”).

4.4.6 Processes with a CLI

A process has many characteristics including whether it is a subprocess or a detached process, whether it interacts with a user or not, and whether it has a CLI to transition between images or has only one image and exits when the image does. Interactive processes interact with users and have a DCL CLI. Batch processes do not interact with users directly and have a DCL CLI. Network processes are started to handle some network request and have a DCL CLI. The Network ACP manages creations of network processes. The job controller process manages creations of most interactive and batch processes. The job controller creates a process 1) to handle unsolicited terminal input, 2) in response to the SUBMIT command, 3) when cards are placed in a card reader, or 4) when the Send Message to Job Controller ($SNDJBC) system service is called. The created process executes the image LOGINOUT, which is an image that is installed with privileges. The following sections describe DCL processing, processing for interrupted DCL images, and LOGINOUT processing for interactive logins and batch jobs.

4.4.6.1 CLI Processing

The only CLI allowed in the evaluated system is the DCL interpreter. Part of LOGINOUT’s initialization processing is to perform a rundown on LOGINOUT so that DCL can execute. One of the first things DCL does is to declare an exit handler that prevents process deletion after an image exit. This allows multiple images to be run one after the other. After initialization, DCL is in a command loop, reading and processing records from SYS$INPUT. Some commands may require execution of an image (e.g., COPY, some SHOW commands) and some commands can be performed by DCL itself.

DCL performs the following steps to process a command requiring image execution: it invokes the $RUNDWN system service to clean up the previous image if necessary, it declares an internal (supervisor mode) routine as an exit handler, it invokes the $IMGACT system service, it establishes data structures necessary (e.g., an argument list) to execute the image, and it calls the image. After image termination, $EXIT calls the exit handler declared earlier and control is transferred back to the CLI loop after cleanup actions.
4.4.6.2 CLI Image Interruption

If a nonprivileged image is interrupted (CTRL/Y or CTRL/C), the image context is saved and the DCL’s main command loop executes. If the next command is one that DCL can perform internally (no image needed), the interrupted image can be continued. Execution of a new command requiring an image destroys the previous image (since the $RUNDWN system service is called). Issuing DEBUG after an interruption causes control to transfer to the debugger. Linking an image with /NOTRACEBACK and /NODEBUG prevents this. All TCB code is linked this way.

If a privileged image is interrupted, DCL saves the current privileges and resets the process privileges to what they were before the image started. Then DCL returns to the command processing loop. All but the ATTACH, CONTINUE, and SPAWN commands cause the image to be run down before execution of the new command. This is also true of images for which a user has execute but not read access.

4.4.6.3 Interactive Processes

When a user is not logged into a terminal, the terminal is protected by the default device protections and can only be accessed by system processes. The ownership of terminals is defined during system initialization. When the RETURN key is pressed at a terminal, the terminal driver sends a message to the job controller via a mailbox (see page 106, “Mailboxes”). The job controller creates a new process with LOGINOUT as the image and associates the process with the terminal. Since the process created is a system process, the association is allowed. LOGINOUT then performs the following security-relevant steps:

- Verifies system password, if enabled
- Prompts for a username
- Verifies that the input device is a terminal
- Prompts for a password and verifies it (It will also prompt for a secondary password if required.)
- Checks for valid login with respect to login qualifiers and account restrictions (see page 172, “Account Restrictions”)
- Changes the process name to the username
- Performs audit, if validation failed
- Records security attributes from the user database in the process data structures
- Initializes the CLI
- Writes the successful login message
- Performs audit for successful login
- Passes control to the CLI.

LOGINOUT is also used for logout operations when the LOGOUT command is used. Security-relevant steps in logout processing are performing audit of a logout, writing a logout message, and requesting $EXIT in executive mode (which invokes the $DELPRC system service). After the process is deleted, the job controller receives control of the terminal.

4.4.6.4 Batch Job Processing

When the SUBMIT.EXE image is activated in CLI (or when the $SNDJBC system service is called), a message is sent to the job controller via a mailbox. The job controller creates a job record and puts it in a
list of pending requests for batch queues. When a batch job completes (or when the number of active batch jobs is below the maximum), the job controller selects an inactive job’s record from the pending request list and creates a process with LOGINOUT as the image. LOGINOUT processing for batch jobs is similar to that for interactive logins. One main difference is that no password checks are done for a job submitted from an executing process. However, SUBMIT propagates a validated username, and user account authorizations are checked before starting the job. For logout operations LOGINOUT writes a logout message to the log file and closes the file. It then requests the $EXIT system service which requests the $DELPURC system service.

For VAX, when cards have been placed in a card reader similar processing is performed. The card reader driver sends a message to the job controller via a mailbox. The job controller creates a process with the image INPSMB (the input symbiont). The input symbiont then reads the username and password from the $JOB and $PASSWORD cards, and performs validation processing similar to LOGINOUT. The input symbiont opens a file in the user’s default directory and reads the rest of the cards into that file. The input symbiont then sends a message to the job controller (via a mailbox) to create a job record for the job just read in. From this point, processing is exactly like the processing for the SUBMIT command. Alpha does not support card readers.

### 4.5 Interprocess Communication

OpenVMS/SEVMS provides a number of mechanisms for processes to communicate among themselves. Mechanisms such as common event flag clusters, mailboxes, logical names and global sections, are not visible cluster wide. Only two mechanisms, DECnet and resource locking, are available across the nodes in a cluster. Global sections are a fast way to communicate, since pages in a process’s virtual address space are shared. Processes must agree on synchronizing reads and writes. See page 89, “Global Sections,” for a description of global sections. The following sections give more details on common event flag clusters, mailboxes, logical names, and resource locks. In addition, DECnet and the OpenVMS/SEVMS utilities that support user-to-user communication (MAIL and PHONE) are described.

#### 4.5.1 Common Event Flag Clusters

Event flags are an OpenVMS/SEVMS mechanism for signaling to a process. A set of system services can be used to set and test event flags. An event flag can be specified as a parameter to a number of system services as a means to signal to the calling process when some system event occurs (e.g., I/O complete, timer expiration).

Flags are grouped into named clusters of 32 flags (bits). For each process, there are two clusters defined that are local to the process (flags 0-31 and 32-63), and two that may be shared among several processes (flags 64-95 and 96-127). Each of the two sharable clusters is a Common Event Flag Cluster (CEFC). The Associate Common Event Flag Cluster ($ASCEFC) and Disassociate Common Event Flag Cluster ($DACEFC) system services are used to create and access common event flag clusters. Each CEFC has a name that includes the GID associated with the process that created the cluster.

The Set Event Flag ($SETEF) and Clear Event Flag ($CLEF) system services provide for the manipulation of event flags within a cluster. The Read Event Flags ($READF), Wait for Single Event Flag ($WAITFR), Wait for Logical Or of Event Flags ($WFLOR), and Wait for Logical And of Event Flags ($WFLAND) provide for reading and testing event flags.
There are two types of clusters: temporary and permanent. A temporary cluster is deleted when the last process associated with it is deleted. Creation of a permanent CEFC requires the PRMCEB privilege. A permanent CEFC not deleted when the process is deleted. A process can delete a permanent cluster by using the Delete Event Flag Cluster ($DLCEFC) system service. If the deleted CEFC has processes associated with it, OpenVMS/SEVMS makes the cluster a temporary cluster until all associated processes are deleted.

### 4.5.2 Logical Names

Within OpenVMS/SEVMS, logical names provide a mechanism for mapping a character string to a replacement string, called an equivalence name. They are used primarily to provide logical representations of devices and filenames. For example, most OpenVMS/SEVMS commands expect to read input from the logical name SYS$INPUT, which is assigned the equivalence name of the terminal device identifier during login. An equivalence name may be a logical name, in which case translation continues until the equivalence name is resolved to a string that is not itself a logical name. A logical name may have multiple equivalence names; the equivalence names are interpreted as a list of logical name tables to be searched when the logical name is translated.

Logical names are stored within logical name tables that are associated with a particular process, job, or group, or are available system-wide. Logical names can be used as IPC because logical name tables can be shared among processes. For example, a logical name might be used to communicate a data value from one application environment to another via a shared logical name table.

Associated with each process is a directory of non-shared logical name tables that are stored in process space. During login, OpenVMS/SEVMS initializes the process directory with the names of default logical tables. When a process is spawned, entries from process-private logical name tables may be handed down from the parent process.

The system directory table contains the names of all logical name tables that are stored in system space (S0) and can be shared among processes. Shareable logical name tables include job logical name tables that can be shared among the processes in a single job, group logical name tables that can be shared by processes having the same associated GID, and a system logical name table that can be shared among all of the processes on the system. Changing the system directory requires SYSPRV privilege; creating logical names in group logical name tables requires the GRPNAM privilege; and creating logical names in the system logical name table requires SYSNAM privilege.

Logical names have an associated hardware access mode. There may be multiple definitions for a logical name in the same logical name table, differentiated only by access mode. During translation, if an access mode is specified by the caller, names at a less privileged access mode are ignored. Unless the caller has the SYSNAM privilege, a logical name with an access mode different than the caller’s access mode cannot be created or altered. The access mode of a name cannot be more privileged that the access mode of the logical name table that contains it.

OpenVMS/SEVMS provides a set of system services to create and delete logical names and logical name tables, and to translate a logical name.
CHAPTER 4. SOFTWARE ARCHITECTURE

4.5.3 Mailboxes

Mailboxes provide synchronization via I/O of information between processes. A mailbox is implemented as a pseudo-device; that is, a mailbox supports virtual I/O for interprocess communication. Messages are read and written with standard I/O mechanisms. (Note that mailboxes are not used for delivery of user mail messages; a mailbox is simply an IPC mechanism.)

Mailboxes are dynamically created by processes and are represented by the same data structures as devices (see page 121, “I/O Management”). The driver is MBDRIVER, which services $QIO system service requests. Messages are stored in non-paged pool and are read in the order that they are written. Only one process can read a message. If there is a write request with no pending read, the data is buffered. If there is a read request with no write, the reading process has the choice to wait or return.

There are two kinds of mailboxes, temporary and permanent. Temporary mailboxes require the TMPMBX privilege to be created and are deleted when there are no more channels assigned to them. Permanent mailboxes can only be created by processes with the PRMMBX privilege and must be explicitly deleted.

The Create Mailbox ($CREMBX) system service is the mechanism to create a mailbox and assign an I/O channel to link the creating process with that mailbox. This system service creates a virtual mailbox device named MBAn, where n is the unit number (which cannot be determined before the mailbox is created). A process creating a mailbox usually requests the creation of a logical name that translates to the mailbox device name. This logical name for the mailbox is associated with a logical name table, one for each type of mailbox (temporary or permanent). Other processes identify the mailbox by its logical name when they assign a channel to it.

4.5.4 Resource Locks

Lock management is used to synchronize shared access to resources based on a resource name. Locking is performed by mutual agreement on resource naming conventions between cooperating processes; a hierarchy of names may be defined. The name of an OpenVMS/SEVMS resource is composed of three parts: a resource domain name, a specified name, and a hardware access mode (optionally, a parent lock instead of a resource domain name may be specified).

Locking is used extensively by OpenVMS/SEVMS to provide concurrency controls for access to system resources. For example, RMS uses executive mode locks to coordinate concurrent accesses to files and records. Applications may define and use locks to synchronize sharing of application-specific data structures and resources.

4.5.4.1 Lock Management System Services

The lock manager is invoked by the Enqueue ($ENQ) and and Dequeue ($DEQ) system services. The lock manager uses an event flag or an AST to signal the status of a requested lock.

Locks can be requested in different modes as listed below:

- Null lock (no access is requested, but the resource data structures are not deleted)
- Exclusive lock
4.5. INTERPROCESS COMMUNICATION

- Protected write (requests write access to the resource, allows sharing with concurrent read mode locks)
- Protected read (requests read access to the resource, allows other readers)
- Concurrent write (requests write access to the resource, allows other writers)
- Concurrent read (requests read access to a resource (locked by protected write), allows other readers (and one writer)).

A lock request on a resource can be in one of three states: granted, in a wait queue or in a conversion queue
(meaning a lock request by that process on the resource has been granted for one mode and is waiting to be
converted to another lock mode).

The $ENQ system service tries to grant a requested lock immediately. If this cannot be done, the lock request
is placed on the resource’s waiting queue. The $ENQ service is also used to perform lock conversions. The
$DEQ system service dequeues a lock or cancels an ungranted lock request. The $GETLK system service enables a process to obtain information about a lock.

4.5.4.2 Lock Management Data Structures

Data structures used in lock management are a Resource Block (RSB), a Resource Hash Table, a Lock Block
(LKB), and a Lock ID table. Figure 4.8 shows the relationships between these data structures.

The RSB is used to maintain all lock management information for a particular resource, such as a resource
name, access mode, pointers to queues for granted, conversion and waiting LKBs, and a 16-byte field called
the value block. The value block associated with a resource may be read by a process when obtaining a lock,
and may be written when releasing a write lock. In this way, data may be passed along with the lock itself.
The RSB for a given resource is located through the Resource Hash Table; the RSB is used to locate all of
the LKBs associated with the resource. The Lock ID table is used to locate all lock blocks on the system.
The LKB is owned by the process requesting the lock service and maintains process-specific information such
as a unique lock ID, the current state of the lock, (possibly) a pointer to a parent lock, and an access mask.

4.5.4.3 Resource Domains

In order to control the sharing of the lock value block of a resource, each resource is associated with a
single resource domain. Each resource domain is defined by a Resource Domain Data Block (RDDB) in S0
space that contains the GID associated with the domain, a pointer to the Object Rights Block (ORB) for
the domain, and a reference count. During login, access to two different resources domains is established
for a process, a system domain, and a default process domain corresponding to the process’s group. A
Resource Domain Access Block (RDAB) defines each resource domain with which a process is associated,
and includes an access mask that specifies the process’s authorized access to the resource domain. The Set
Resource Domain ($SET.RSDM) system service can be used to access additional resource domains. Figure
4.8 describes the data structures associated with resource domains and their relationship to the other lock
management data structures.

When a root lock (no parent specified) is initially created, the $ENQ system service assigns the lock to one of
the resource domains associated with the process, depending on the parameters specified by the caller. The
access mask from the RDAB is copied into the LKB for the lock. For subsequent requests for resource locks,
the requesting process must have the required access to the resource domain as defined in the lock access
Figure 4.8. Lock Management Data Structures
mask. However, when a process is executing in a protected access mode, resource domain access checks are not performed.

4.5.4.4 Distributed Lock Management

Management of locks in a VMSCluster environment is distributed. One node in the cluster is designated as the director node for a set of resources. All other nodes contain identical directory vectors pointing to the director node. When a node wants a lock on a resource, it sends a message to the director node designated in the directory vector asking for the master of the resource.

The director node contains a Resource Hash Table for all resources in the cluster. When it receives a request for the master of a resource, it finds (via the Resource Hash Table) an RSB corresponding to the requested resource. It then checks the RSB for the master node (if the directory flag is set) or declares itself as the master (if the directory flag is clear). The director node then copies the RSB to the requesting node with the master node’s ID.

A master node has a Resource Hash Table designating a master RSB for all resources it manages. When it receives a request for a lock on a resource, the master node grants the lock (if possible), copies the master RSB into a process RSB on the requester node, places its ID and the lock ID in the process RSB, and places the requesters ID and the lock ID in the master LKB (the master LKB is on the master node). Figure 4.9 shows the links between the nodes for distributed lock management.

4.5.5 DECnet

DECnet provides a message-passing interface for communicating between two processes on different nodes. A process on a local node initiates a DECnet connection request specifying the DECnet Task application (see page 85, “Network ACP”) and a specific DCL command file name. On the remote node, DECnet creates a process and executes the command file. (See page 174, “Authentication for DECnet Applications”, for a description of how the security attributes of the created process are determined.) Once the DECnet connection is established, standard I/O services (using DCL commands, high-level programs, or OpenVMS/SEVMS system services) are used to read and write messages.

4.5.6 MAIL and PHONE Utilities

OpenVMS/SEVMS provides a MAIL utility for sending and receiving mail messages and a PHONE utility for interactive communication between users.

Each OpenVMS/SEVMS user is registered in the MAIL utility database. This database contains the default directory in which the user’s mail files are located and other user-defined mail parameters. A user may view and change the user’s own entry in the mail database; a user with SYSPRIV privilege may view all of the entries in the database. Also, a user with SYSPRIV privilege can delete or change arbitrary entries in the database. The MAIL utility provides a set of system services, as well as DCL commands, to deliver mail to OpenVMS/SEVMS users and to maintain the mail database.

The PHONE utility is a DCL command that allows a user to communicate interactively with another user, either locally or at a remote DECnet node. Intranode conversations take place via mailboxes created by the
Figure 4.9. Distributed Lock Management
PHONE utility; for internode phone conversations, the DECnet Phone application is used.

The PHONE utility is a privileged image; it is installed with OPER, WORLD, NETMBX, PRMMBX, and SYSNAM privileges. The OPER privilege is used to establish a connection to a local terminal being phoned; the WORLD privilege is used to establish a remote connection using the DECnet Phone application; the NETMBX privilege is also needed to communicate using DECnet; the PRMMBX privilege is used to create the mailboxes used for local communication; and the SYSNAM privilege is used to declare system logical names for mailboxes used by the PHONE utility;

When the PHONE utility is invoked, a connection is established between the initiator and the receiver(s). The PHONE utility takes control of the terminal of the initiating user and forces the screen to a specific format. It also creates a mailbox or a DECnet connection to facilitate the transfer of messages. At the receiver's end, a message is displayed to the terminal informing of the incoming call. The receiver has to initiate the phone conversation by invoking the PHONE utility command and typing "ANSWER." Any user can turn off the PHONE facility by setting BROADCAST off.

4.5.6.1 MAIL and PHONE Utilities in SEVMS

To send mail on SEVMS, the sender must be logged in at a label that is lower than or equal to the classification associated with the receiver's default mail directory (or must have BYPASS or DOWNGRADE privilege). The MAIL utility will "write up" to the recipient's directory. To read mail, the user must be at the same label as that of the default mail directory. The B1 Security Features User's Guide (SFUG) recommends that the user set the label of the mail directory to the maximum authorized classification of the user.

The use of the PHONE utility in SEVMS is constrained by MAC. The labels on the mailbox (for local conversations) or on the DECnet connection (for internode conversations) determine whether any actual conversation can take place. These controls prevent any potential downgrade of information. When used over DECnet for internode conversations, the PHONE utility constrains the initial message displayed to the receiving user to a single fixed format containing only the username of the user initiating the PHONE conversation.

4.6 File Management

The OpenVMS/SEVMS file system performs several functions. It maintains the directories on a volume; provides the functions for operating on files such as open, close, create, delete, extend, and truncate; manages free volume space; maintains file integrity; maps logical blocks to virtual blocks; and translates requests from applications into those for device drivers. In OpenVMS/SEVMS, file management is primarily performed through a set of system services collectively known as RMS, though files can be directly manipulated through the $QIO system service. $QIO is discussed on page 125, "I/O Processing".

This section discusses the format of disk files and volumes, the Extended QIO Processor (XQP)(which actually supports the file system structure), tape files and volumes, the backup utility, VMScluster file access, RMS (a set of services layered on top of the $QIOservices), RMS journaling, and OpenVMS/SEVMS features for distributed transaction management.
4.6.1 Disk File Specification

In OpenVMS/SEVMS, a file is a set of disk blocks, each of which is of 512 bytes long. A file specification has six components: node, device, directory, filename, type, and version. The node refers to the particular cluster machine on which the file resides. The device is the name of the associated physical device on the cluster node.

The following is an example of a local file specification:

\[\text{[DIRECTORY]} \text{MYFILE.DAT;}\text{10}\]

An example of a specification for a file on a cluster-visible volume is:

\[\text{NODE1\$DU1;}\text{[DIRECTORY\_SUB]} \text{MYFILE.DAT;}\text{10}\]

Note the node name, “NODE1,” and the device specification, “\$DU1.” Also note that the subdirectory “SUB” is specified as part of the pathname. Finally, an example of a remote file specification using DECnet is listed below.

\[\text{NODE2::DISK1;}\text{[DIRECTORY2]} \text{MYFILE.DAT;}\text{12}\]

Access to the file listed above is obtained through the DECnet FAL application.

A directory is simply a file that contains pointers to other files. For each file contained in the directory, the filename, type, version, and a pointer to a file header, called a file identifier, are listed. The file header contains the security attributes of the file and the location of the file on disk. Reference to a file can be made using either a file identifier or a pathname containing directory references.

Each disk contains a Master File Directory (MFD) that contains a list of all the files on the disk, including the directories and the MFD itself. Each directory may contain subdirectories, to a maximum of seven levels of subdirectory.

File types (filename extensions) are primarily used to distinguish files associated with a particular utility, such as mail or an editor. The extensions provide a method of default file specification for certain utilities, which assume a file type when a filename is given without an extension. However, the file type in and of itself doesn’t guarantee whether the file is, for example, an object or editor text file. The only case in which a particular extension is required is in the case of directory listings, which must have “.DIR” as the type and a version number of one.

File version numbers differentiate revisions of a file. Version numbers range from 1 to 32,767. By convention, when a file is modified, OpenVMS/SEVMS creates an updated version. The version number is incremented by one each time a new version of the file is created.

4.6.2 Disk Volumes

For disks, a file structure called Files-11 On-Disk Structure (Files-11, for short) is imposed by the XQP.\(^1\) Files-11 refers to the logical ordering of disk structures from the lowest level, block, to the highest level, volume (disk) sets.

\(^1\)RMS uses the XQP.
Files are organized on volumes or volume sets. A volume is a disk. A volume set is one or more disks that are treated as one volume by OpenVMS/SEVMS. Only one directory structure exists in a volume set. Two types of volume sets exist: tightly coupled and loosely coupled. Tightly coupled volume sets are self-identifying, in that each disk in the set has the same structure (volume set) name, but each has a different volume name. The first volume in the set contains a file that holds a list of the volumes (volume labels) in the set. By contrast, loosely coupled volume sets are not self-identifying. That is, there is no file that lists the volumes that belong to the set. In this case, it is the responsibility of the operator and application program using the volume set to maintain the correct order of the volumes for files that span across volumes. Both types of volume sets allow multivolume files. However, only one file per volume is allowed to span to another volume in a loosely coupled volume set.

### 4.6.2.1 Volume Structure

Each volume contains an index file, which is the root of the entire disk file structure. The index file contains the information used to identify and access the volume. The index file contains the boot block, the home block, the file headers for all of the files on the disk, and the allocation bitmap. If the volume is one that may be used to boot the system, the boot block contains a small program that reads the operating system into memory when the boot block is read and executed by the hardware bootstrap of the machine. If the volume is not bootable, the boot block contains a program that outputs a message to the system console indicating that the volume does not contain a boot program. The home block is always the second (first good writable) block on a disk. There are several copies of the home block stashed on the disk so that inadvertent destruction of the home block still allows XQP the capability to locate other files on the volume. The home block contains the volume name, volume protection (both MAC (for SEVMS only) and DAC), the UIC of the owner, the name of the owner, the maximum number of files that can be present on the volume at any one time, and other information such as that needed to locate the rest of the index file. The MOUNT command (and system service) uses a valid home block when invoked to mount a volume. Thereafter, the XQP locates files on the volume and the manipulates other data on the disk.

### 4.6.2.2 Block Structure

Files are viewed by the file system as being a contiguous set of virtual blocks. The file system maps virtual block numbers into a logical block numbers for a volume. The disk driver translates the logical volume block number into a cylinder/track/sector address for a physical block on disk. A block is 512 bytes in size, regardless of how it is viewed. This is the smallest unit that may be transferred from disk to memory. A contiguous group of logical blocks is called a cluster (not to be confused with a VMScluster, described on page 57, "VMScluster"). A disk cluster is the basic unit of space allocation on the volume. The cluster size is determined either by default or by specifying it when initializing the volume (via the DCL INITIALIZE command). The cluster size is used in allocating new and extended files. Cluster sizes are limited by the size of the disk. Smaller disks are usually assigned smaller cluster sizes; larger disks usually have larger cluster sizes to avoid overhead during space allocation. Two clusters are not necessarily contiguous.

A file’s data blocks consist of one or more extents. An extent is a set of contiguous clusters that are allocated to a file. The number of extents of a file is an indication of how fragmented the file is. A contiguous file is one whose data blocks comprise a single extent.
4.6.3 File Identification

Each file in a volume set has an associated 48-bit binary value called a file identifier, or file ID, supplied by the file system when the file is created. Normally, a directory is used to associate a file name with a file ID; however, the use of a directory is not required. The file ID points to the location of the file header.

The file header is not actually part of the file. The header is contained in the volume index file. The file header contains all the information needed to access the file, including the list of extents that describe where the file data is physically located on the volume. Multiple file headers are used for files with a large number of extents, as well as for files that span volumes or have a very large Access Control List (ACL).

The file header is divided into six areas, five of which are variable in size. The areas are named: header, ident, map, ACL, reserved, and end checksum. Only the header area and the end checksum are required. The header area contains variable-length fields but the overall size is fixed at one block (512 bytes). The end checksum is a fixed-size (two byte) area that contains an additive checksum of all the other words in the header block. The checksum is verified each time the file header is read and recalculated each time the file header is written. The header area contains, among other items, the file owner and the discretionary and mandatory protection fields. In OpenVMS, the mandatory protection field is not allocated when files are created, except when the CLASS_PROT SYSTEM parameter is set. In the latter case, OpenVMS allocates a zero-filled area for the mandatory protection field. The mandatory protection field is ignored by OpenVMS when processing existing files.

The ident area stores accounting information, such as creation date and time, revision count, expiration date and time, and backup date and time. The map area contains a list of pointers describing the extents allocated to the file. The pointers are ordered according to the virtual blocks that are represented. Each Access Control Entry (ACE) in the ACL describes the discretionary protection associated with the file. The ACL area may span multiple headers, occupying the ACL area in each header. The reserved area is available for user applications and exists only in the first header (for files that require multiple file headers).

4.6.4 Extended QIO Processor (XQP)

File I/O is performed by a combination of the $QIO system service, the $XQP, and other system software (e.g., device drivers). This section describes the functions provided by the XQP. The $QIO system service and device drivers are discussed in more detail on page 125, "I/O Processing". The $QIO service handles the I/O request preprocessing, an example of which is parameter parsing. Hardware-dependent functions and I/O postprocessing are performed by other system software. The XQP performs most of the rest of the file I/O functions, such as access control (see page 149, "Files" for details), file open (or any file request not involving a transfer of data), bad block handling that occurs during an I/O request, and file transfer requests in the circumstance where there is not enough information in memory to convert the virtual blocks of a file to the physical blocks of the disk. The XQP also handles the file highwater mark (the largest numbered relative block ever written to the file) and file system synchronization.

The file system uses a variety of mechanisms to serialize activity. Spin locks and elevated IPL are used to serialize access to the I/O database (see page 121, "I/O Management"). Locks are used to synchronize access to disk blocks and shared file system data structures.

The buffer cache is a set of systemwide buffers allocated from the paged pool. Each VAX node has an I/O

---

2 The CLASS_PROT parameter is normally set to zero for OpenVMS and always set to one for SEVMS.
buffer cache maintained by the XQP. All XQP I/O is performed using the buffer cache. The buffer cache contains file structure components such as headers, bitmaps, and directories. Access to these structures is coordinated through locks that correspond to the associated files. In fact, each data block in the buffer cache is identified by the lock under which a file operation is to take place. The lock is based on a file ID.

The XQP is a special case of an entity, called an ACP, that performs I/O support processing associated with a device (ACPs are discussed on page 126, “Ancillary Control Processes”). In most cases, ACPs are independent system processes. The XQP is not a separate process; it is contained in an image that is mapped into a process’s P1 space when a process is created. A permanent global section is created for the image during system initialization.

The XQP cannot be called directly by a user. The interface to the XQP is the $QIO system service. The XQP is activated by a kernel mode AST.

The primary data structure involved in an I/O operation is the I/O Request Packet (IRP). The IRP contains the parameters necessary to perform the I/O operation. Information in the IRP is supplied by $QIO system service and device driver routines. IRPs make up the device queues. $QIO uses the IRP and other I/O data structures to determine whether a driver, the XQP, or a special routine is to process a particular function.

The XQP allocates a portion of a process’s P1 space, called impure storage, used to contain several data items for exclusive use by the XQP during I/O processing. The most significant contents of the impure storage area: a private per-process kernel stack, an internal queue, and a per-process data area, a portion of which is used to save context.

The XQP is entered via a dispatcher that is part of the XQP image. The dispatcher is called via ASTs that are sent from one of two routines. The first sends a special kernel mode AST that ensures that the current IRP will not be invalidated by process deletion between the end of I/O post processing and the time a normal kernel mode AST could be delivered to the process. The second routine sends a “normal” kernel mode AST to the XQP. The address of the XQP dispatcher is contained in the AST sent from either one of the routines. The AST parameter sent to the XQP is the IRP.

When the XQP AST executes, the XQP image is entered from one of three routines. The first calls the XQP to perform ACP I/O operations and to provide needed mapping information to map virtual block read and write operations into physical I/O. The second calls the XQP to handle bad blocks as they are encountered and to provide mapping information for lengthy virtual I/O operations. The third calls the XQP to queue a deaccess operation on an idle channel.

The XQP delays a file transaction under certain circumstances, such as a resource wait (e.g., a lock request), a cache wait, or a wait I/O completion. The XQP then switches execution from its internal stack to the kernel mode stack and returns to the caller. When the XQP executes after the resource has been made available or the pending I/O request completes, then the XQP switches context back to its internal stack to handle queued operations. ASTs are used to resume the XQP. The XQP also handles cleanup after a file operation. Cleanup includes writing modified buffers to disk and purging and deallocating associated data structures.

If an error occurs, the XQP uses one of three methods to reflect the condition back to the user. Status may be returned directly through $QIO. Alternatively, the error condition may be returned in a field in the IRP through the use of a macro routine. The second method is used for status that indicates success or is informational to the user. For fatal errors, XQP executes a CMNU instruction that causes an exception to be generated. The exception handler saves the associated code that becomes the error status. The error status is stored in the IRP and reflected back to the caller through normal I/O postprocessing.
When the XQP completes processing of all pending requests and the cleanup operations have been performed, the synchronization locks held by the XQP are released. XQP then posts I/O completion status for the file system request to an I/O post-processing routine. If the request involved a transfer of data, then a software interrupt is requested to be executed on the primary CPU (this is to guarantee that transfer operations complete in the proper order on a multiprocessor system). However, some file functions are processed by direct calls to specific I/O post-processing routines from XQP. If the function is not done in the XQP but rather in a device driver or Function Decision Table (FDT) routine, then a software interrupt is posted and the IRP for the request is inserted at the end of the I/O post-processing queue. When the XQP gets control back from the I/O post-processing routines, it calls a routine that checks whether a volume was marked for dismounting. The XQP also calls the appropriate auditing service calls if file system auditing is enabled.

### 4.6.5 Tape Volumes

The file storage system for magnetic tapes uses labels corresponding to the American National Standards Institute (ANSI) format. Files-11 does not apply to tapes. Tape records are ordered sequentially. The details of the ANSI structure are presented in [27]. The labels have been designed to include MAC information for SEVMS. Each OpenVMS/SEVMS tape volume contains the following sections: beginning-of-tape, volume label, file header labels, file data, file trailer labels, and the end-of-tape marker. The file header labels, file section, and file trailer labels are separated by tape marks. The logical end of a volume is indicated by two consecutive tape marks.

#### 4.6.5.1 Volume Labels

There are two volume labels. The first volume label is 80-characters in length and describes the label type, name, and owner of the volume. Only four fields of the volume label are accessible to the user: the volume identifier, accessibility field, implementation identifier, and owner identifier. The volume identifier is a six-character field containing the volume name, and is established when the volume is created. The accessibility field is a one-character field available for installation functions to interpret. This field can be used to indicate whether other users have access to the tape volume, have read or write access, or have access based on the OpenVMS/SEVMS DAC volume protection (the default). The implementation identifier is used to indicate the application that wrote the tape. The implementation identifier influences how certain fields are interpreted on the tape. The owner identifier field is available to the user. The field is generally used to process tapes that are to be interchanged with another Digital operating systems. The owner identifier field is not involved in checking access to the tape volume.

The second volume label contains the volume-owner field. This field is used to store DAC volume protection. The protection attributes are assigned to a tape when the volume is initialized. There is no DAC information kept for each file on the tape. The DAC for each file is the same as that of the volume.

SEVMS uses the International Organization for Standardization (ISO) user-format labels to implement an additional volume header label and an additional volume trailer label. The third volume label contains the minimum and maximum classification of the tape volume. The trailer label is the same as its header counterpart with the exception of the characters “EOV” that appear in the label identifier. A trailer label is written on the tape volume for each header label. End-of-volume labels are used to indicate that a file that could not be written entirely on one volume continues on the next volume of the set.
4.6.5.2 File Labels

The header area of a file contains up to four file-header labels for each file on the tape volume. Two of the header labels are optional. The first file header label includes the file identifier and the creation and expiration dates. An accessibility field is also included in the first header label and is used similarly to the volume accessibility field. The second header label describes the record format (fixed or variable size) of the file, including the maximum record size and the maximum block size. The third header label is usually present, but is optional. It contains the RMS attributes of the file, including the internal record format (sequential, relative, or indexed). The fourth header label merely contains an extension of the file name if it does not fit into the first header label.

There is also an end-of-file trailer label. It is the same as the file header label with the exception of the characters “EOF” that appear in the label identifier and a block count indicating the number of data blocks in the file section. End-of-file labels are used to delimit a file that terminates on the volume.

SEVMS uses the ISO user-format labels to implement an additional file header label. The fifth file header label contains the file classification. The fifth end-of-file label serves the same purpose as OpenVMS trailer labels, except that it contains file MAC information. The OpenVMS/SEVMS version of the BACKUP utility provides support for handling MAC label information.

4.6.5.3 Tape I/O Operations

As mentioned previously, tapes are accessed only sequentially. Multivolume files and tape volume sets are similar in implementation to loosely coupled disk volume sets. One other notable difference between tape and disk processing is that block sizes are determined by the user with tapes; with disks, the block size is fixed.

Virtual I/O operations to tape are handled by the magnetic tape ACP. The operations performed by the tape ACP include: create, access (check file access and open file if access checks are passed), deaccess (close), delete, mount, and control functions used to position the tape. A different version of the ACP is installed with SEVMS. The SEVMS tape ACP provides access controls based on file classification.

4.6.5.4 The Backup Utility

On an OpenVMS/SEVMS system, tape volumes are used primarily for backup processing. The BACKUP utility is part of the TCB because it is responsible for maintaining tape header and trailer labels. For BACKUP, all tape volumes are mounted as “foreign.” This means that the BACKUP utility, not the magnetic tape ACP, processes the tape header and trailer labels. Consequently, BACKUP enforces the OpenVMS/SEVMS security policy by controlling mandatory and discretionary access to the contents of tape volumes during save and restore operations.

Single or multiple disk files may be saved or restored using BACKUP. A file specification for BACKUP can include all of the elements of a Files-11 file, except for a node name. Multiple files may be saved into a single entity, called a save-set. A save-set is treated as a Files-11 file and may be either a disk file or a tape file. Node names may be used in save-set specifications. BACKUP treats all tape files as save-sets. Since only one of the input and output specifiers may be a save-set, no tape-to-tape operations may be performed through BACKUP. One or more specific files may be restored from save-sets through the use of a qualifier to
the BACKUP command. Alternatively, an entire directory tree, including subdirectories, may be restored through BACKUP.

In SEVMS, BACKUP is installed with DOWNGRADE and UPGRADE privileges because it performs its own MAC checks on tape files. If a user has DOWNGRADE and UPGRADE privileges or the BYPASS privilege, the BACKUP utility can be used to mount a volume with non-Digital volume labels, bypass MAC access controls, or initialize unlabeled tape volumes.

4.6.6 Accessing Files In A Cluster

Files may be accessed on a remote cluster node through two different methods. The methods are distinguished primarily by whether the file resides on a cluster-visible volume or not. If the file resides on a cluster-visible volume, the MSCP protocol is used to access the file; otherwise, DECnet is used.

4.6.6.1 Access to Cluster-visible Files

The MSCP protocol is the main mechanism used to perform I/O on files on a cluster-visible volume. MSCP-served disks and dual-pathed disks are cluster-visible. HSC disks are automatically cluster-visible. Local node disks are not visible cluster-wide unless they are specifically set up as such using the MSCP server. To make a disk cluster-visible, the MSCP server must be loaded (specified by SYSGEN) and certain parameters must be set either via SYSGEN or through a data file.

4.6.6.2 Access to Local Disks on Cluster Nodes

For disks that are not cluster-visible, the DECnet FAL application is used to obtain access. DECnet is invoked when a full pathname is specified for a file that includes the DECnet node name. A full pathname may also include an explicit access control string that is composed of a username and a password. Access to the file is mediated on the remote node based on the security attributes of the remote FAL process. The security attributes are established as described on page 174, “Authentication for DECnet Applications”.

4.6.7 Record Management Services

RMS is a set of system services that provides support for file management and maintenance based on records. RMS can be invoked directly, through a high-level language, or through a run-time library. RMS supports disk and tape storage management primarily. However, I/O to devices such as terminals and printers is also supported. RMS provides the capability to perform block I/O (for file or record organizations defined by the user). RMS also supports file sharing at the record level using the distributed lock manager.

4.6.7.1 File Organization

Three types of file organizations are supported by RMS: sequential, relative, and indexed. A sequential file is one in which records are placed in order one after the other. A relative file is one in which all records are
assigned a number that indicates its position in the file with regard to the first record. An indexed file is one in which its records can be retrieved either randomly or in order using a key embedded inside each record.

In a sequential file, records are arranged in the order in which they were stored into the file. In general, records in a sequential file must be accessed in order starting at the beginning of the file, although RMS supports random access of a sequential file with a fixed-length record format. Random access of a variable-length record sequential file can be achieved by supplying RMS with the record file address that is returned to the calling program by RMS each time the record is read or written. Generally, records can only be added to the end of the file (i.e., after the most recently added record). The sequential file organization is supported for all devices and is the only file organization supported for devices other than disks.

Sequential and random access of records is supported in relative files. The record position is used by RMS to calculate the record's physical location on the disk. In relative files, each record is assigned to a cell of equal length. The actual length of the record within a cell may vary, up to a maximum of the size of the cell.

Similar to relative files, indexed files can be accessed both sequentially and randomly, and are supported only on disk devices. An index structure that contains a key that determines the order of records in the file is used to store and retrieve data from the file. The physical placement of each record is controlled by RMS.

The logical EOF is maintained by RMS; therefore, the logical EOF and physical EOF may be different.

4.6.7.2 Record Formats

There are four different record formats supported by RMS on OpenVMS/SEVMS: fixed length, variable-length, variable-length with fixed-length control, and stream. The lengths of the records used with the fixed and variable-length formats are defined by the user. A file with variable-length records with fixed-length control contains records that vary in size, but are preceded by a control field that is of the same size for every record. Records with stream format are delimited by special characters. Stream records are interpreted as a continuous set of bytes. The most common delimiters are the carriage return and line feed characters. The stream record format is supported only for sequential files on disk devices and for terminal I/O operations.

RMS runs in executive mode (and uses executive mode ASTs) and is a part of the TCB. File access control is performed in the XQP. However, RMS provides an additional interface to the XQP to allow DAC information to be read or modified. RMS services also support I/O to remote nodes through the DECnet Data Access Protocol (DAP) (which is transparent to most application programs). RMS also controls record sharing within files; the XQP controls sharing at the file level.

RMS coordinates access to files through the use of hardware protection modes and the distributed lock manager. RMS provides hardware access mode protection for some services and their associated memory (e.g., process local and global I/O buffers) using executive mode locks. The access mode of most RMS services is checked before any of the service functions are performed. Some data structures are protected using executive mode read and write access. They include those data structures and associated I/O buffers used for RMS operations that are RMS-controlled, process-permanent, and image-activated. In addition, the internal RMS I/O buffers and buffers containing information related to index files are protected with user mode read and executive mode write access.

---

3 There are four services for which the access mode checks are not considered relevant and for which access-mode protection is not verified (because they require the file to be closed to perform their respective operations): enter (create a directory entry), erase (delete a file and its directory entry), remove (delete a directory entry), and rename.
The RMS services are divided into two general groups: those that support files, and those that support records. The file services perform functions such as the following: create, open, extend disk space, close, and get file characteristics (via ACP and XQP functions). The following are examples of the functions provided by the record services: get, find, put, update, delete, and operations not related to record I/O, such as associating certain methods of access (i.e., record streams) with an open file.

RMS can be accessed through assembly macros (OpenVMS/SEVMS MACRO language). A high-level language called File Definition Language (FDL) is available that can be used through a defined set of utilities to perform RMS services. Much accessing of RMS is done via the run-time libraries that are associated with the various language compilers; therefore, RMS functions are transparent to most application users.

The next lower level of the I/O mechanism is the $QIO system service. RMS is essentially an interface to $QIO. RMS converts its input into input for $QIO. Unprivileged users may also call $QIO directly. The $QIO system service is explained on page 121, "I/O Management".

### 4.6.8 RMS Journaling

RMS journaling refers to a capability used to protect the integrity of data in files on OpenVMS/SEVMS. There are three types of RMS journaling: after-image, before-image, and recovery unit. After-image journaling allows the caller to redo the last set of RMS operations. After-image journaling is designed to protect against disk crashes and hardware failures. Before-image journaling provides the capability to undo the last set of RMS operations on the file; this helps prevent bad or erroneous data from entering the file. Recovery unit journaling allows the caller to specify a set of operations that must be done in their entirety or not at all. Recovery unit journaling prevents a set of operations from being completed if they are interrupted due to a system crash or an error. Any or all three journaling mechanisms may be used on a file. The OpenVMS/SEVMS TCB does not use RMS journaling for any TCB files.

Journaling may be used for any RMS file to be updated. However, journaling cannot be used for non-RMS file operations; nor can it be used for files not updated in place (i.e., files for which a new version is created each time a write occurs).

If journaling is applied to a file, a continuous record is kept of all modifications made to the file. The RMS Recovery Utility is used to apply (in the case of after-image journaling) or undo (in the case of before-image journaling) RMS operations. Recovery units, called transactions, are defined through system services supported by DECdtm.

### 4.6.9 Transaction Management

OpenVMS/SEVMS has an integrated facility, DECdtm, that provides support for distributed transaction management using the two-phase commit protocol. Application programs can use the transaction management system services to coordinate updates to "resources" that may be located on different system nodes. The transaction manager uses the IPC services to communicate with the resource managers involved in the transaction. When an application commits or aborts a transaction, the transaction manager notifies all of the resource managers and coordinates the global commit or abort. Each resource manager must be capable of aborting and recovering the update operations that were performed by the transaction on its resources. In OpenVMS/SEVMS, RMS is the only resource manager in the evaluated configuration; updates to multiple files can be coordinated using the transaction manager.
4.7 I/O Management

The I/O management functions of OpenVMS/SEVMS are performed by device drivers and their associated data, by device independent service routines within the OpenVMS/SEVMS Executive, and by several system services that handle requests from other system services and directly from user images. This section provides an overview of the four major components of OpenVMS/SEVMS I/O management: the I/O Data Structures, I/O System Services, device drivers, and I/O support processes (ACP). In addition, the OpenVMS/SEVMS queue management functions and the SEVMS print symbiont are described.

4.7.1 I/O Data Structures

The OpenVMS/SEVMS I/O Database contains a number of data structures that are used to communicate among the various parts of the I/O subsystem. In general, the data structures are stored in non-paged system space (S0). Device registers are part of the physical I/O address space of VAX processors and can be accessed by directly by VAX instructions (see page 32, "I/O Management"). Device registers are mapped into S0 space during system startup, and the appropriate data structures are allocated and initialized for all physically attached devices. In a VMScluster system, the CONFIGURE process communicates with the other nodes of the cluster and integrates any cluster-visible devices into the I/O database.

Figure 4.10 provides an overview of the information contained in the structures within the I/O Database. At the center is the Unit Control Block (UCB) that is defined by the system for each device. The UCB defines the current state of the device, and points to a number of other data structures. A queue of pending I/O Request Packets (IRPs) for the device is maintained with the listhead in the UCB. During I/O interrupt processing, the current context (registers and PC) for the device driver is maintained within the UCB. This supports the use of the "fork dispatcher" to reduce the interrupt level for further processing after the initial hardware interrupt has been handled (see page 25, "Interrupt and Exception Processing"). Sufficient context is maintained within the UCB to resume I/O processing at a lower IPL.

The Device Data Block (DDB) contains data common to a specific type of device, including the location of the associated device driver tables. The Channel Request Block (CRB) and the Interrupt Dispatch Block (IDB) describe the state of the device controller, and point to the relevant control blocks for the I/O bus adapter to which the controller is attached.

The other data structures within the I/O database provide external entry points for each device driver, entry points for controller and device unit initialization routines, interrupt dispatching vectors for I/O service routines, addresses of device registers, and bit maps describing the allocation of datapaths. In a VMScluster, the I/O database on each node includes data structures for the devices that are cluster-visible.

A Channel Control Block (CCB) provides the logical connection between a process and a device. Unlike the other I/O data structures, CCBs are located in a table in P1 space for each process; the maximum number of CCBs that can be allocated is controlled by a SYSGEN parameter. Each CCB points to the UCB for the device that has the channel assigned to it by the process. A process may have multiple channels assigned to the same device.
Figure 4.10. I/O Data Structures
4.7.2 System Services

The I/O interface supported by the TCB includes systems services to perform basic I/O functions. The primary I/O system services perform device allocation and deallocation ($ALLOC, $DALLOC), establish the logical connection between a process and a device ($ASSIGN, $DEASSIGN), queue requests for device I/O ($QIO, $QIOW), and cancel outstanding I/O requests ($CANCEL).

The primary I/O system service is the Queue I/O Request ($QIO), which is discussed in detail on page 125, “I/O Processing”. The $QIO system service call contains parameters that define the logical I/O channel and the I/O function being requested. Optional parameters specify an event flag to be associated with the I/O, a storage location for the I/O completion status, and/or an address of an AST procedure to be called when the I/O completes.

4.7.3 Device Drivers

Device drivers provide the interface between the $QIO system service and device-specific functions. A driver is composed of several distinct components, and may contain an internal VMS interface that differentiates between a class driver and a port driver. This section describes device drivers in general and then discusses several of the OpenVMS/SEVMS drivers.

4.7.3.1 Device Driver Components

A device driver module is made up of three tables (Prologue Table, FDT, and Driver Dispatch Table (DDT)) and a set of routines that actually perform the I/O processing. The driver prologue table contains identifying information needed by OpenVMS/SEVMS to load and initialize the driver.

The Function Decision Table lists all of the valid function codes for the driver and the address of the driver’s I/O pre-processing routine for each function. OpenVMS/SEVMS supports a standard set of function codes, divided into three categories of I/O: physical, logical, and virtual. Physical I/O function codes include operations such as seek and write-physical-block. Logical function codes include operations such as rewind and write-logical-block. Virtual I/O function codes are valid only for file-structured devices (see page 111, “File Management”), and include operations such as create-file and write-virtual-block (a virtual block is relative to the beginning of a file). The function decision table defines the function codes that a particular device driver supports; in effect, the device driver defines its own set of operations.

The Driver Dispatch Table contains the entry points for all of the standard driver routines. Each driver contains a start-I/O routine, an interrupt service routine, and, optionally, routines for unit initialization, controller initialization, timeout handling, canceling I/O, and dumping registers (for error diagnostics).

The driver executes in kernel mode; however, the context in which a driver routine executes is dependent upon the specific routine. The pre-processing routines (in the FDT) execute in process context. The driver routines in the DDT (e.g., start-IO) execute in system context using the interrupt stack (see page 25, “Interrupt and Exception Processing”).

During I/O processing, OpenVMS/SEVMS provides services to device drivers to synchronize and coordinate the driver activities with other system activities. The OpenVMS/SEVMS interrupt service routine known as the fork dispatcher maintains a queue of interrupted driver fork processes. The fork dispatcher is the interrupt
service routine for a set of interrupt priority levels that are used to complete I/O interrupt handling at IPLs that are lower than the device interrupt priority level. The fork dispatcher dequeues a driver fork process, restores the driver’s context from information in the device UCB, and places it into execution.

4.7.3.2 Class Drivers and Port Drivers

For some types of devices, the device driver functionality is split between two different types of drivers: a class driver and a port driver. A class driver encapsulates the generic functionality of the device type while a port driver addresses the physical characteristics of a specific hardware device and/or interface medium. A single class driver can be associated with multiple port drivers to support different hardware implementations (e.g., for the terminal driver); alternatively, a single port driver can be associated with multiple class drivers (e.g., to support devices on the Small Computer System Interface (SCSI) bus). In a VMScluster, the MSCP interface is supported by disk and tape class drivers (DUDRIVER and TUDRIVER). DECnet within a VMScluster is supported by the CNDRIVER class driver. Different port drivers are used within a VMScluster to implement MSCP for different hardware interfaces (e.g., PADRIVER for a CI interface, PEDRIVER for an NI interface).

4.7.3.3 Volume Shadowing Driver

In addition to the hardware-supported volume shadowing available with the HSC (see page 58, “HSC”), OpenVMS/SEVMS provides software-supported volume shadowing. The software supported volume shadowing (also called host-based volume shadowing) is provided by a device driver (SHDRIVER) that manages a single virtual unit and redirects the I/O operations to one or more of the devices with volumes that are members of the volume set.

Figures 4.11 and 4.12 illustrate the basic procedures involved in servicing read/write I/O requests for software-supported volume shadowing. For a read I/O request, the SHDRIVER, after receiving an IRP from the $QIO system service, generates a clone of the IRP and passes it the disk driver for the physical device that the shadow driver determines can return the requested data in the shortest possible time. For a write I/O request, the SHDRIVER generates one clone IRP for each member of the shadow set and passes the clone IRPs to all of the device drivers associated with the volume set.
4.7.4 I/O Processing

This section describes the basic processing that occurs for a typical I/O request. The specific steps are outlined, and the relevant system and driver components are described.

1. A process invokes the $ASSIGN system service to assign a channel as the logical connection between the process and a device. If the device is non-sharable, the $ALLOC system service would be invoked prior to the $ASSIGN in order to allocate the device to the process.

2. A process requests I/O by invoking the $QIO system service. The channel number as well as a function code must be specified.

3. The $QIO system service performs device-independent validity checking on the input parameters, initializes the IRP, and then calls the driver’s preprocessing routine for the specific function.

4. The driver’s preprocessing routine completes the I/O preprocessing by checking the validity of the device-dependent $QIO parameters and adding device-specific information to the IRP. The driver then calls an OpenVMS/SEVMS service routine to queue the IRP for the device. The service routine raises the IPL to the driver’s fork IPL (and, in a multiprocessor environment, obtains the spinlock) prior to transferring control to the driver’s start-I/O routine. If the device is busy, the service routine queues the IRP to the device UCB, rather than directly transferring control.

5. The driver’s start-I/O routine invokes OpenVMS/SEVMS service routines to prepare for the data transfer and then to activate the device. Preparations include obtaining controller access, initializing transfer data areas, and locking the device registers (raising IPL to device IPL and obtaining the spinlock). For standard VAX I/O, the device is activated by storing device-specific function code information in the Control and Status Register (CSR) for the device (in I/O space). With mailbox I/O, the driver activates the device by loading the address of the mailbox containing the I/O command into the internal mailbox register (which activates the I/O Port Module (IOP) module). After activation, the driver releases the device lock and invokes an OpenVMS/SEVMS service to wait for the device interrupt. The OpenVMS/SEVMS service routine saves the context of the driver fork process and suspends its execution.

6. When a device interrupt occurs, the driver interrupt service routine obtains control through the device vectors within the SCB at device IPL. The device interrupt service routine reads the device register
or mailbox) to obtain the status information. The driver handles the interrupt and then invokes an OpenVMS/SEVMS service routine to reschedule the driver code at fork IPL for further I/O completion processing.

7. When dispatched by the fork dispatcher, the device driver executes at fork IPL to perform device-dependent I/O post-processing. When this processing is complete, the driver calls an OpenVMS/SEVMS service routine to perform device-independent post-processing.

8. After all of the system context post-processing is complete, the OpenVMS/SEVMS service routine queues a kernel mode AST to the process that originally requested the I/O. When delivered, the kernel mode AST executes in process context to transfer status and data, and, if requested, to signal (via a user mode AST or event flag) the I/O completion to the process.

There are two strategies employed by device drivers for data transfer: direct I/O and buffered I/O. With direct I/O, the user mode pages in the process context are locked in physical memory for the duration of the I/O. For buffered I/O, pages of non-paged pool are allocated for the data transfer, and the kernel mode AST copies the data to the process buffers. Direct I/O is generally used by drivers for Direct Memory Access (DMA) devices, while buffered I/O is generally used for programmed I/O devices (where the data transfer to memory is performed by the driver not the device).

4.7.5 Ancillary Control Processes

An Ancillary Control Process (ACP) can be employed to extend the functionality of a device driver to accomplish device-independent tasks. OpenVMS/SEVMS uses ACPs to support the file system (Files-11 On-Line Disk Structure (ODS)-2 XQP and the magnetic tape ACP) and to support network connections (network ACP). While most ACPs are implemented as separate processes, the XQP is mapped into the P1 space of individual processes.

There is a set of function codes that are recognized by the $QIO system service as ACP function codes. These codes are used to create, access, deaccess, and delete objects (files and network connections), and to control and modify the attributes of objects. The file system ACP (XQP) performs operations that are ancillary to the actual I/O operations: directory lookup; file allocation, extension, and deallocation; and reading or changing the attributes of files. For ACP function codes, the $QIO system service queues the IRP to the ACP’s queue, rather than to the device driver’s queue. The file system ACP is discussed on page 114, “Extended QIO Processor (XQP)”; the magnetic tape ACP is discussed on page 117, “Tape I/O Operations”; and the network ACP is discussed on page 85, “Network ACP”.

4.7.6 Device Types

In addition to actual I/O devices, OpenVMS/SEVMS provides several types of pseudo-devices: mailboxes, non-local cluster-accessible devices, network devices, spooled devices, pseudo-terminals, virtual terminals, and remote terminals. For each of these pseudo-devices, a UCB is created when the pseudo-device is created. For some devices, a template UCB is cloned by the device driver to create a pseudo-device.

Mailboxes are the primary OpenVMS/SEVMS IPC mechanism (see page 106, “Mailboxes”). Non-local cluster-visible devices are disk and tape devices that are managed locally by class drivers communicating to other nodes that physically access the device (see page 85, “The MSCP Servers”). A network device is
created as the result of a DECnet reference to a device on a different node. For a spooled device, I/O is redirected to an intermediate disk device; the physical device is managed by an output (or input) symbiont, as described in the next section on queue management.

Virtual terminals facilitate the reconnection of a user’s session to a different physical device after communication has been interrupted. Remote terminals represent a login connection via the DECnet SET HOST command, usually, though not necessarily, from a different node. Pseudo-terminals can be used to emulate terminal interactions with users.

4.7.7 Queue Management

The OpenVMS/SEVMS queue manager supports the submittal of batch jobs and print requests. The queue manager interacts with job controller processes to receive requests and with print symbiont processes and job controller processes to dispatch requests. A batch request is implemented by a detached process that executes the submitted command procedure. A print request causes the requested files to be printed on a device managed by a print symbiont. A queue entry is the control information and attributes associated with each request.

The queue manager uses the $IPC system services to communicate with the job controller and print symbiont processes. In a VMScluster, each node has a job controller and may have zero or more print symbiont processes. By default, there is one queue manager per cluster; however, there may be up to five queue manager processes in a cluster. The job controllers on each node communicate with one another to determine which node(s) should run the queue manager. If a node fails, the other nodes will determine, based on system parameters, the node(s) that should start up queue manager processes.

The queue manager database resides on a cluster-visible volume and is accessible to all nodes. The database consists of three files. A master file contains permanent information—definitions of forms and printer characteristics, list of queues, names of the queue managers, and the names of nodes eligible to run queue managers. A queue file contains the job entries currently in the queue. A journal file contains recovery and status information to allow the queue manager to be restarted on a different node if necessary. If there are multiple queue managers, there will be one master file but multiple queue and journal files, one for each queue manager. While active, a queue manager maintains an in-memory database that contains all of the information necessary to service requests.

4.7.7.1 Types of Queues

There are three types of queues: execution, device, and generic.

Execution queues hold batch requests. A batch request is defined by a DCL command procedure and associated job control information. A request may be held in a queue until a specific time, and batch jobs may synchronize with each other. The queue manager dispatches batch requests by communicating with the job controller on the node selected to run the batch job. The queue manager controls the number of requests active from any given queue; but the execution of the jobs is controlled by the OpenVMS/SEVMS scheduler, just as for any other process.

Device queues hold print requests. A print request is defined by a set of file names to be printed, and associated printer control information. The queue manager dispatches a print request by giving the files and
control information of a request to a print symbiont (via the job controller). A print symbiont is a system process that is associated with a number of devices. The symbiont actually performs the I/O operations to print the files.

Generic queues can be defined that refer to multiple device or execution queues. For example, a generic queue can be used for the submittal of print requests to be printed on any one of a number of printers associated with different device queues.

4.7.7.2 Print Symbionts

A print symbiont is a process that manages a print request from the queue manager by performing I/O to a printer owned by the symbiont. A standard print symbiont is included with OpenVMS, and SEVMS additionally includes a print symbiont that labels output with human-readable sensitivity labels.

OpenVMS/SEVMS allows the system administrator to specify templates for the print job header and trailer pages. The templates contain directives to describe the layout of these pages (for example, where to put the username on the pages). As a part of the template, a unique random number can be printed on both the header and trailer pages. By default OpenVMS/SEVMS print symbionts print the same random number on the header and trailer pages of each print job. There is also a set of system-wide logical names that can be defined to further control the printing of the header and trailer pages. The administrator can use the OpenVMS/SEVMS device control library to automatically send a reset sequence to the printer prior to sending the file to be printed.

4.7.7.3 SEVMS Print Symbiont

The SEVMS print symbiont is designed to control and format labeled printed output for printers that are directly connected to the system. SEVMS provides the system administrator with a set of print symbiont templates (including a default template) to format the labels being printed. SEVMS selects a print symbiont template for a print job based upon file classification, printer type, and printer width.

The SEVMS print symbiont labels each page of output unless the PASSALL option is specified in the print request. When printing a file, the SEVMS print symbiont builds an image of each page and analyzes all the characters prior to passing them to the printer. Unless the PASSALL option has been specified, any character or control sequence that will not produce results known to be consistent with the page image will not be printed. If the PASSALL option is specified, the action taken by the print symbiont is determined by system administrator parameter settings. The symbiont may 1) ignore the option and perform as normal, 2) allow the option and pass all characters and control sequences to the printer, or 3) abort the print job. The use of the PASSALL option is auditable.

The SEVMS print symbiont has a feature that makes it compatible with the previous versions of SEVMS. If a system administrator activates this feature, the print symbiont does not guarantee page label printing integrity by analyzing the characters to be printed. However, the use of the compatibility feature is excluded from the evaluated configuration.
4.8 System Operation

The security of OpenVMS/SEVMS relies on proper system administration. A number of features help the administrator ensure the correct operation of the system. The following sections describe the administrative controls provided by OpenVMS/SEVMS, the administrative system consoles, some system integrity checks, system startup, and system installation.

4.8.1 Administrative Control

A large set of privileges is provided by OpenVMS/SEVMS. As described on page 138, “Privileges”, only a subset of these privileges are security-relevant. Any user who is granted any of these security-relevant privileges is an administrator. However, the granularity of privilege allows system administrative personnel to be granted limited control without requiring that all operators be given complete administrative control of the system. Many privileges are used only by trusted software, for system software debugging, hardware maintenance, or for special circumstances.

4.8.1.1 Administrative Roles

Four particular combinations of privileges provide access to system functions that are daily administrative functions needed by specific personnel. These groupings are a useful way to group privileges, but are not intended to imply that OpenVMS/SEVMS separates these roles. Any user granted security-relevant privileges is outside the scope of the enforced policy and may be able to use the granted privileges to gain more privileges.

Operator An administrator with OPER and MOUNT privileges may perform operator functions. Specific examples of operator actions include starting and stopping printers, receiving operator messages, mounting devices with special characteristics, and setting and overriding login limits and times. With the addition of the IMPORT, UPGRADE, and DOWNGRADE privileges, an operator may relabel data and import unlabeled data in SEVMS.

Group Administrator An administrator with GROUP, GRPPRV, and GRPNAM may control processes in the same group, read or modify user records of group members, control objects owned by users in the same group, act as a group operator, and mount and access group devices.

Auditor An administrator with the SECURITY privilege may control the system auditing capabilities, display audit data as it is generated, analyze audit data, and monitor the evasion of suspected break-in attempts. An administrator acting as an auditor would also require discretionary access to the system audit log to analyze audit data. This access could be either from DAC protections on the audit log file or from additional privileges.

System Administrator An administrator with the SYSPRV and SYSNAM privileges may act as the system administrator. This administrator is capable of adding and deleting users, mounting volumes for
system-wide access, manipulating the mail database, and controlling objects. In addition, the administrator may manipulate the security databases, including the security and integrity labels in SEVMS.

4.8.1.2 System Account

Every OpenVMS/SEVMS system has a system account that has all privileges, and is designed to bypass some account restrictions. It is used to run many trusted processes, and may be used by administrators to gain access to the system when failures result in other accounts being inoperative. Many of the special bypasses of the system account operate only from a CPU node console.

4.8.2 System Consistency Checks

A VMScluster configuration relies on inter-node communication to maintain consistent state information. If the communication hardware fails in certain specific ways it could cause two independent subclusters to boot and interfere with each other in catastrophic ways. To prevent, this a system parameter is used to control the number of systems expected in the VMScluster. If the number falls to half or less of this number, the VMScluster connection manager on each node causes the node to wait until another system boots. Therefore, a large number of concurrent hardware failures may stop the execution of the entire VMScluster. This may be corrected by bringing the nodes down from their consoles, and booting them with parameters entered from the console.

A serious malfunction will cause a VAX node to shut itself down. It will then send a message to any other nodes in the cluster with information helpful to the other nodes in recovering a consistent system state. If an administrator requests a VAX node shutdown, the system will shutdown processes in an orderly fashion, flush its caches, and then execute a standard crash routine.

OpenVMS/SEVMS is capable of recovery from power failures if the memory has battery backup. This is accomplished by saving critical state information in the system restart parameter block. When power returns, all device drivers are signaled that there was a power failure; this allows them to reinitialize if necessary.

4.8.3 System Startup

A stand-alone node not participating in a VMScluster system may save a copy of memory and startup by loading that copy of memory and signaling all device drivers and system processes to initialize. This startup method uses the same basic mechanisms as the power failure recovery. Only designated processes may be running when memory is saved; and the process used to capture memory is logged out by the system before saving the memory image. This process facilitates fast booting of a stand-alone node.

A system boot relies heavily on the system consoles to boot the nodes. As discussed on page 48, “Console Subsystems”, one of the key differences between various models of VAX processor is the style of system console that is used. The consoles range from the VAX CPU executing ROM bootstrap code, to an entirely separate VAX CPU dedicated to system console operations. The differences for purposes of booting may be summarized as the location of the primary bootstrap program (VMB), the functionality of the program loading VMB, and the method of determining the system device from which to boot VMS. Regardless of its implementation, the console subsystem initializes the VAX CPU, runs basic integrity tests, locates physically contiguous good memory, loads a VMB image into that memory, and executes it.
If a VAX node has no local boot device (i.e., a satellite workstation), the boot ROM sends a broadcast message over the cluster communication device. A node connected directly (or via the CI) to the boot device responds, after confirming the satellite’s hardware address as a member of the cluster. The node then uses DECnet (see page 86, “Maintenance Operation Protocol”) to send the VMB to the satellite node.

If the VAX node is not a satellite, it loads VMB from a local device. The boot device that is configured by VMB may be either a local disk or a disk managed by an HSC node.

VMB executes with the memory management hardware disabled. VMB sizes memory, sets up device I/O structures for the boot device, and loads the secondary bootstrap program (SYSBOOT). SYSBOOT reads the system generation parameters and lays out the system virtual address space based on those parameters. It then loads the system base image SYS.EXE and a number of the system loadable images listed on page 243, “VMS Executive Images”. SYSBOOT transfers control to the routine EXE$INIT in the image EXEC$INIT.

EXE$INIT enables the memory management hardware, loads the rest of the loadable images, initializes the scheduler and I/O databases, starts other CPUs on this bus, waits to hear from the critical VMScluster nodes, and transfers to the scheduling routine.

The scheduler schedules the SWAPPER process to run. The SWAPPER process runs its initialization routine and creates the SYSINIT process. The SYSINIT process executes the DCL command file STARTUP.COM. This process creates the system processes such as OPCOM, the job controller, the SMISERVER, and the DECnet processes. STARTUP configures the I/O database, and installed images specified by the VMSIMAGES.DAT file. Finally, the startup process executes a series of site-specific command files and enables interactive logins.

### 4.8.4 System Installation

Installing OpenVMS and SEVMS is a comparatively easy operation. The applicable installation guide presents to the administrator, command by command, the instructions to boot a processor from installation media. The installation routine that actually builds the system on the system disk requires that the passwords be changed on the accounts that are shipped with the system. The guide continues for installing OpenVMS onto a common cluster system disk. After that installation procedure has run its course, if SEVMS is being installed, the installation guide for that product instructs the administrator through adding to the newly built OpenVMS system.

After installing the software, the cluster configuration utility can be used to add the other nodes into the cluster; and the network configuration procedure can be used to configure the network with minimal effort. OpenVMS/SEVMS is capable of tuning itself to the local hardware configuration and local load conditions with the AUTOGEN utility. The AUTOGEN utility utilizes the SYSGEN and SYSMAN utilities to set system parameters to reasonable values. An administrator may use SYSGEN directly to override AUTOGEN’s tuning decisions.
Part II

Security Functionality
Chapter 5

TCB Protected Resources

This chapter describes the OpenVMS/Security Enhanced VMS (SEVMS) subjects and objects and their relevant security attributes. The role of the security attributes during access mediation is presented in Section 6.1, “Access Mediation”, beginning on page 155.

5.1 Subjects

A subject in OpenVMS and SEVMS is a process. An untrusted subject is a process executing in user mode, without security-relevant privileges and without a User Identification Code (UIC) that is associated with the SYSTEM category of users. The OpenVMS/SEVMS privileges are listed in Tables 5.1 and 5.2 beginning on page 138. The SYSTEM category of users is described on page 137, “User Identification Code”.

Processes have a number of security-relevant attributes associated with them. Most of these security attributes are stored within the Access Rights Block (ARB) data structure. The attributes and data structures are described below.

5.1.1 Access Rights Block

There is a one-to-one correspondence between an ARB and an OpenVMS/SEVMS process; the ARB is pointed to by the process’s Process Control Block (PCB). An ARB describes the security profile of the process (see Figure 5.1). An ARB contains the user’s UIC, several Rights Lists (identifiers that the process holds), a Mandatory Access Control (MAC) classification block (user clearance), and a working privilege mask. Each of these components is described in detail in subsequent sections.

5.1.2 User Name

Each user authorized to access the OpenVMS/SEVMS system is assigned a user name by a system administrator (using the AUTHORIZE command). The user name is the unique index into the User Authorization File (UAF), and provides access to the security attributes that are permanently stored. During login processing, these attributes are used to initialize the ARB data structure. The user name itself is stored within the Job Information Block (JIB) associated with the process in P1 space. If auditing for the user has been enabled, an audit flag in the PCB is set.
<table>
<thead>
<tr>
<th>Working Privilege Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flags</td>
</tr>
<tr>
<td>Classification Block</td>
</tr>
<tr>
<td>Process Rights List Pointer</td>
</tr>
<tr>
<td>System Rights List Pointer</td>
</tr>
<tr>
<td>Extended Rights List Pointer</td>
</tr>
<tr>
<td>Image Rights List Pointer</td>
</tr>
<tr>
<td>Reserved</td>
</tr>
<tr>
<td>UIC</td>
</tr>
<tr>
<td>Local Rights List</td>
</tr>
</tbody>
</table>

Figure 5.1. The Access Rights Block
5.1.3 User Identification Code

OpenVMS/SEVMS assigns each process the UIC that corresponds to the user name given during login. The UIC is stored with the user name in the UAF. It consists of both a User Identifier (UID) and a Group Identifier (GID). The group identifier determines the group to which the user belongs. For UIC-based discretionary access (see page 156, “UIC-Based DAC”), a subject is in one of four categories (SYSTEM, OWNER, GROUP, or WORLD) determined by comparing the subject’s UIC to the owner UIC of the object. Certain GIDs are administratively defined to be in the SYSTEM category of users and have extended access to TCB objects.

OpenVMS/SEVMS allows several user names to correspond to a single UIC. The OpenVMS/SEVMS Trusted Facility Manual (TFM) states that, for the evaluated configuration, every UIC must be associated with only one user name.

5.1.4 Identifiers and Rights Lists

A subject’s Rights List is made up of identifiers and associated attributes that are held by a user. The identifiers held by a subject determine the subject’s access rights to objects that have Access Control List (ACL) entries specifying any of the identifiers held by the subject.

Identifiers are defined in the Rights Database by a security administrator using the AUTHORIZE utility. An identifier is held by a user if that user’s UIC is associated with the identifier. This association information is also kept in the Rights Database. A subject’s Rights List is initialized during process creation based on the information in the Rights Database. The subject’s Rights List is composed of several different lists, as defined by the four Rights List pointers in the ARB. Conceptually, however, there is a single Rights List for a subject.

If a user holds an identifier with the DYNAMIC attribute, then the user may dynamically include or exclude that identifier from the subject’s Rights List using the SET RIGHTS LIST command. A subject’s Rights List can also change dynamically when the process executes an image that is part of a protected subsystem. A protected subsystem image file contains a SUBSYSTEM Access Control Entry (ACE) that defines a set of identifiers and attributes associated with the subsystem (see page 157, “The Access Control List”). The subsystem identifier will be added to the subject’s Rights List only for the duration of execution of the subsystem image. When the image terminates, its subsystem identifiers are removed from the subject’s Rights List.

5.1.5 SEVMS Subject Classification

During SEVMS login processing, a classification block is associated with a process and stored in the ARB.\footnote{Refer to “Labels” on page 159 for a description of the classification block.} The classification associated with a process remains constant for the duration of the process. Within the UAF for each user, there is a maximum clearance, a minimum clearance, and a default clearance. If a user specifies a clearance during login, it must be within the range specified in the UAF; if no clearance is specified, the user’s default is used as the subject’s classification.
5.1.6 Privileges

Privileges can modify the access mediation behavior of the Trusted Computing Base (TCB) and authorize the use of privileged TCB functions. Some system services require that a process hold a specific privilege to invoke the service. The active set of privileges associated with a process is represented by the working privilege mask in the ARB and is maintained redundantly by the TCB in the working privilege mask in the Process Header (PHD). The mask in the process header exists because some older system services and privileged images check this mask, rather than the mask contained in the ARB.

5.1.6.1 Privilege Categories

OpenVMS supports 36 privileges that Digital groups into six different categories based on the potential compromise that may result if the privilege was abused or misused. The six categories are as follows:

- Normal - there are no compromises
- Group - potential to affect objects and processes with the same UIC group
- Devour - potential to consume non-critical system-wide resources
- System - potential to interfere with normal system operation
- Object - potential to compromise object security
- All - potential to control the system.

SEVMS supports three additional privileges (DOWNGRADE, UPGRADE, and IMPORT) that affect the mandatory access control policy, resulting in a total of 39 privileges. Table 5.1 describes the OpenVMS/SEVMS privileges that are not security-relevant; Table 5.2 describes the security-relevant privileges. Any process having a security-relevant privilege is a trusted subject since it can affect the secure operation of the TCB. The applicable TFM describes each privilege and the specific capabilities that it authorizes.

5.1.6.2 Privilege Propagation

Within the UAF record for a user is a default privilege mask and an authorized privilege mask. Each bit within a mask indicates that the user possesses a specific privilege. During login, the default privilege mask

<table>
<thead>
<tr>
<th>Category</th>
<th>Privilege</th>
<th>Activity Permitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>NETMBX</td>
<td>Create network connections</td>
</tr>
<tr>
<td></td>
<td>TMPMBX</td>
<td>Create temporary Mailbox</td>
</tr>
<tr>
<td>Devour</td>
<td>ACNT</td>
<td>Disable accounting</td>
</tr>
<tr>
<td></td>
<td>ALLSPOOL</td>
<td>Allocate spooled devices</td>
</tr>
<tr>
<td></td>
<td>BUGCHK</td>
<td>Make bugcheck error log entries</td>
</tr>
<tr>
<td></td>
<td>EXQUOTA</td>
<td>Exceed disk quotas</td>
</tr>
<tr>
<td></td>
<td>PRMCEB</td>
<td>Create/delete permanent common Event Flag Clusters</td>
</tr>
<tr>
<td></td>
<td>PRMGBL</td>
<td>Create permanent Global Sections</td>
</tr>
<tr>
<td></td>
<td>PRMMBX</td>
<td>Create permanent Mailboxes</td>
</tr>
<tr>
<td></td>
<td>SHRMEM</td>
<td>Create/delete structures in shared memory</td>
</tr>
</tbody>
</table>

Table 5.1. OpenVMS/SEVMS Non-Security-Relevant Privileges
<table>
<thead>
<tr>
<th>Category</th>
<th>Privilege</th>
<th>Activity Permitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>GROUP</td>
<td>Control processes in the same group</td>
</tr>
<tr>
<td></td>
<td>GRPPRW</td>
<td>Group access through SYSTEM protection field</td>
</tr>
<tr>
<td></td>
<td>GRPNAM</td>
<td>Insert logical names in a group logical name table</td>
</tr>
<tr>
<td>Devour</td>
<td>ALIPRI</td>
<td>Set base priority higher than allotment</td>
</tr>
<tr>
<td></td>
<td>AUDIT</td>
<td>Access system security audit file</td>
</tr>
<tr>
<td></td>
<td>OPER</td>
<td>Perform operator functions</td>
</tr>
<tr>
<td></td>
<td>PSWAPM</td>
<td>Change process swap mode</td>
</tr>
<tr>
<td></td>
<td>WORLD</td>
<td>Control any process</td>
</tr>
<tr>
<td></td>
<td>SECURITY</td>
<td>Perform security related functions</td>
</tr>
<tr>
<td></td>
<td>SYSLCK</td>
<td>Lock system wide resources</td>
</tr>
<tr>
<td>System</td>
<td>DIAGNOSE</td>
<td>Diagnose devices</td>
</tr>
<tr>
<td></td>
<td>IMPORT</td>
<td>Mount a non-labelled tape volume</td>
</tr>
<tr>
<td></td>
<td>MOUNT</td>
<td>Execute mount volume QIO</td>
</tr>
<tr>
<td></td>
<td>READALL</td>
<td>Possess read access to everything</td>
</tr>
<tr>
<td></td>
<td>SYSGBL</td>
<td>Create system-wide Global Sections</td>
</tr>
<tr>
<td></td>
<td>VOLPRO</td>
<td>Override volume protection</td>
</tr>
<tr>
<td>Object</td>
<td>BYPASS</td>
<td>Disregard protection</td>
</tr>
<tr>
<td></td>
<td>CMEXEC</td>
<td>Change to executive mode</td>
</tr>
<tr>
<td></td>
<td>CMKRNL</td>
<td>Change to kernel mode</td>
</tr>
<tr>
<td></td>
<td>DETACH</td>
<td>Create detached processes of arbitrary UIC</td>
</tr>
<tr>
<td></td>
<td>DOWNGRADE</td>
<td>Write to a lower secrecy object or lower an object secrecy label</td>
</tr>
<tr>
<td></td>
<td>LOGJO</td>
<td>Issue logical I/O requests</td>
</tr>
<tr>
<td></td>
<td>PFNMAP</td>
<td>Map to specific physical pages</td>
</tr>
<tr>
<td></td>
<td>PHYSJO</td>
<td>Issue physical I/O requests</td>
</tr>
<tr>
<td></td>
<td>SETPRIV</td>
<td>Enable any or all privileges</td>
</tr>
<tr>
<td></td>
<td>SHARE</td>
<td>Access devices allocated to other users</td>
</tr>
<tr>
<td></td>
<td>SYSNAM</td>
<td>Insert logical names in the system logical name table</td>
</tr>
<tr>
<td></td>
<td>SYSPRIV</td>
<td>Access objects through SYSTEM protections field</td>
</tr>
<tr>
<td></td>
<td>UPGRADE</td>
<td>Write to a higher integrity object or raise an object integrity label</td>
</tr>
</tbody>
</table>

Table 5.2. OpenVMS/SEVMS Security-Relevant Privileges
is copied to the process-permanent privilege mask (in P1 space) and to the working privilege mask (in both the ARB and the PHD). The authorized privilege mask in the UAF is copied into the authorized privilege mask in the PHD. Refer to Figure 5.2.

When the TCB tests for a privilege, the check is made against either of the working privilege masks; the TCB assures that these two masks are always equal to one another. During image activation (e.g., during activation of a Digital Command Language (DCL) command), privileges may be inserted in the known image privilege mask, if the image was installed with privilege. Figure 5.3 describes the image activation process as follows: (1) copy the images's privileges to the known image mask in the PHD, (2) OR the known image mask with the process permanent mask, and then (3) store the result in the working privilege mask.

At image rundown, the privileges acquired are relinquished by overlaying the working privilege mask by the process-permanent privilege mask (see Figure 5.4).

Normally, process privileges are bounded by privileges in the user’s default privilege mask or in the known image privilege mask. Users can also activate other privileges defined by the authorized privilege mask by using the SET PROCESS/PRIVILEGES command or by a call to the SETPRIV system service. Figure 5.5 illustrates the processing when the working privileges are changed: (1) OR the authorized and known image masks; (2) AND the result with the privileges to be set; (3) OR the result with current working mask; and (4) store the result into the current working mask. The process-permanent privileges may also be changed; however, the known image privileges are not included in the processing, only the authorized privileges. A process with the SETPRIV privilege can acquire any other privilege, unconstrained by the authorized privilege mask.
5.1. SUBJECTS

Figure 5.3. Image Activation Privilege Processing

Figure 5.4. Image Rundown Privilege Processing
5.1.7 Propagation of Process Security Attributes

When a process is created (with the $CREPRC system service) the security attributes of the new process are determined based on the attributes and privileges of the creating process and the parameters to $CREPRC. When an untrusted subject creates a process, the parameters to $CREPRC are constrained by the security attributes of the creator. The UIC of the new process is the same as that of its creator. The process-permanent privilege mask is the logical AND of the creator’s working privilege mask and the requested privilege mask. The ARB, the authorized privilege mask, and the Rights Lists are copied from the creating process. If the creating process has the SETPRV privilege, then the privileges of the new process can be explicitly set as parameters to the $CREPRC system service.

For detached processes, the UIC of the detached process may be different than the caller’s UIC if the creating process has the DETACH or CMKRNL privilege. If the creating process has the SETPRV privilege, then the working privilege mask of the creating process is not used to restrict the requested privileges.

In SEVMS, the classification label associated with the process is always the same as the creating process’s label, unless the creating process has both the BYPASS and DOWNGRADE privileges and the process being created is a detached process. In this case, the creating process may specify the classification label to be associated with the new detached process.

When a remote process is created by the DECnet software on behalf of a local process, the security attributes of the remote process are determined by the attributes of the local process, user-specified attributes, and the attributes of the DECnet application on the remote node. Section 6.3.7 on page 174 describes the authentication mechanisms used by DECnet. The OpenVMS/SEVMS TFM provides guidance on defining
DECnet applications to conform to the requirements of the evaluated configuration.

In SEVMS, processes created for a remote DECnet application are created with the same classification label as the local process. Application processes that support multiple DECnet connections can only be used by local processes with the same classification label, unless the application process has the BYPASS or DOWNGRADE privilege. In this case, any local process can connect to the remote application process.

5.2 Objects

This section describes the classes of objects that are protected by the OpenVMS/SEVMS TCB. Objects in OpenVMS/SEVMS are the data containers whose contents can be read and/or written by multiple untrusted subjects and for which OpenVMS/SEVMS provides a programming or DCL interface. Objects include volumes, files, common event flag clusters, global sections, resource domains, queues, and logical name tables. Devices are also protected objects; the physically attached devices supported by SEVMS are limited to disk, tape, printer, terminal, and workstation display devices. Except for resource domains, the objects protected by MAC are the same as the objects protected by Discretionary Access Control (DAC). For resource domains, DAC is provided for locking operations and access to lock value blocks, while MAC is enforced only for access to lock value blocks. This section first describes the security attributes and operations that are common to all OpenVMS/SEVMS objects, and then describes each distinct type of object and its security-relevant attributes.

5.2.1 Common Object Attributes

All of the OpenVMS/SEVMS objects are protected based on a common set of security attributes. This section briefly describes each of the attributes and the mechanisms for establishing them. The role that each attribute plays in access mediation is described in more detail in the discussion of protection mechanisms (beginning on page 155, “Access Mediation”).

5.2.1.1 Object Rights Block

Throughout OpenVMS/SEVMS, a standard data structure, the Object Rights Block (ORB), is used to store the protection data for an object. For permanently stored objects (e.g., disk files) the content of the ORB is derived from information that is stored along with the object’s data. For memory-resident objects (e.g., Logical Name Tables) the content of the ORB is maintained as a data structure in system space (S0). Figure 5.6 illustrates the ORB data structure. The ORB contains a reference to the owner of the object (a UIC or an identifier), protection fields for UIC-based access (see page 156, “UIC-Based DAC”), Access Control List (ACL) pointers (see page 157, “The Access Control List”), and, for SEVMS, maximum and minimum classification labels for the object. For file objects, the minimum and maximum labels are identical; however, for all other objects, a range may be specified. The SEVMS TCB supports classification ranges for disk, tape and spooled printer devices, and for volumes and queues. The use of ranges on other types of objects is not supported in the evaluated configuration.
<table>
<thead>
<tr>
<th>Owner UIC</th>
<th>Mutex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flags</td>
<td>Type</td>
</tr>
<tr>
<td>Update Seq. No.</td>
<td></td>
</tr>
<tr>
<td>Mode Protection Vector</td>
<td>System Protection Field</td>
</tr>
<tr>
<td>ACL Queue Forward Link</td>
<td>ACL Queue Backward Link</td>
</tr>
<tr>
<td>Object Type</td>
<td>Name Length</td>
</tr>
</tbody>
</table>

Figure 5.6. The Objects Rights Block
5.2.1.2 Access Types

Discretionary access to objects is controlled by access types that are defined for each class of object. The specific discretionary access types are described below for each object class. For example, the access types of READ, WRITE, EXECUTE, and DELETE are supported for file objects, while the access types for Common Event Flag Clusters are ASSOCIATE and DELETE.

All object classes support the CONTROL access type, which is necessary for changing the protection data (i.e., the ORB) of an object. A subject in the OWNER or SYSTEM category of users for the object (see page 156, “UIC-Based DAC”) is authorized for CONTROL access by default.

In SEVMS, the mandatory access types are defined as OBSERVE, for any access that reads or observes data within an object, and ALTER, for any access that changes the data within an object. To change the protection data of an object in SEVMS, both discretionary CONTROL access and mandatory OBSERVE and ALTER access are required.

5.2.1.3 Access Modes

Many of the system services that provide access to OpenVMS/SEVMS objects support an access mode parameter that associates a hardware access mode with accesses to the object. For example, an access mode may be specified to the Create Mailbox ($CREMBX) system service. The OpenVMS/SEVMS TCB uses the access mode to limit access based on the hardware access mode of the process attempting to access an object. In order to specify an access mode more privileged than user mode, the process itself must be executing in a more privileged mode. Therefore, an untrusted subject cannot specify an access mode other than user mode. The TCB uses these access modes to protect some TCB resources that are implemented as OpenVMS/SEVMS objects. The access mode checks are implemented by the TCB software that manages the specific objects.

In addition, the ORB contains an access mode protection vector; however, the TCB does not use the information for its access mediation decisions. For all OpenVMS/SEVMS objects, a flag is specified in the ORB that indicates that the access mode protection vector in the ORB is invalid.

5.2.1.4 Establishing Object Security Attributes

When an object is initially created, its discretionary access attributes are either explicitly set or are derived from defaults. In general, UIC-based protection fields may be specified to the system service that creates an object, and an ACL can be added after object creation.

Digital provides a set of default security profiles that can be modified by a system administrator; the default profiles are used whenever discretionary protection data is not explicitly specified at object creation. A security profile includes both UIC-based protection fields and an ACL. The profiles supplied by Digital include only UIC-based protection fields; however, an administrator can add an ACL to the security profile. The specific security profiles supplied by Digital are discussed with the other security-relevant topics for each class of object. For files, security profiles are not used; however, several other mechanisms can be used to establish the initial discretionary security attributes for files (see page 149, “Files”).

The owner of an object is generally set by the TCB to the UIC of the subject creating the object. However,
an identifier may be specified as the owner for some types of objects (files and directories) to facilitate the management of disk space. If a subject holds an identifier with the RESOURCE attribute (see page 156, “Identifiers”) then the subject is authorized to create objects with that identifier as the owner. For all physically attached devices, the owner is set to the SYSTEM UID when the device object is created. For GROUP logical name tables and resource domains, the owner is set to the GID of the subject, with a UID of zero.

In SEVMS, the mandatory access classification labels for single-level objects (both maximum and minimum) are always set equal to the subject’s classification label when the object is created. For ranged objects, the maximum and minimum labels are set equal to the subject’s classification label, except for volume objects. During volume initialization, the labels for the volume default to the maximum and minimum classification labels authorized for the username associated with the subject; alternatively, the subject may specify the maximum and minimum (constrained by the user’s authorization range).

5.2.1.5 Viewing and Changing Object Security Attributes

The Get Security Information ($GET_SECURITY) system service can be used to retrieve the security attributes of an object. A subject must have discretionary CONTROL or READ access to use the $GET_SECURITY system service. In SEVMS the subject must additionally have OBSERVE access. The $SHOW_SECURITY DCL command provides a command-level interface to the $GET_SECURITY system service.

Once the initial attributes have been established, the Set Security Information ($SET_SECURITY) system service is used to change the security attributes. CONTROL access is required to change the security attributes. For SEVMS, both ALTER and OBSERVE access are required in addition to CONTROL access. The $SET_SECURITY DCL command provides a command level interface to the $SET_SECURITY system service, and the ACL Editor provides a convenient interface to the $SET_SECURITY system service for maintaining the entries in an ACL.

In SEVMS, unless the subject has the BYPASS or DOWNGRADE privilege, the minimum classification can only be increased. In order to set or change the classification of an object to a range, the SECURITY privilege is required.

Before changing the classification label of an object, the $SET_SECURITY system service assures that there are no active accesses to the object. Depending on the object type, locking or other TCB mechanisms are used to assure tranquillity. Discretionary attributes of an object may be changed even when there are current accesses to the object; the new discretionary attributes are effective for all subsequent access mediation decisions.

5.2.2 Device Objects

Each device within OpenVMS/SEVMS has an ORB that is pointed to by its associated Unit Control Block (UCB). The ORB data structures for physically attached devices are allocated during system startup. The devices are owned by the SYSTEM user and are initialized with the discretionary security attributes of the profile associated with the device type. In SEVMS, the classifications are assigned during SEVMS startup using standard DCL commands. Cluster-visible devices are handled differently; their security attributes are centrally maintained and are automatically established during system startup on each node of the cluster. Additional physical devices may be added or removed after system boot by an administrator; the ORB for
a physical device is not removed even though it may be taken offline.

The ORB data structure for a template device (e.g., Ethernet device) is cloned along with the UCB when the device is created. The ORB data structure for other types of pseudo devices (e.g., mailboxes) are initialized by the TCB using either explicitly set attributes or the security profile for the device type. A mailbox device is created with the $CREMBX system service; its associated data structures are deleted when there are no longer any channels assigned to it (temporary mailbox) or when explicitly deleted using the $DELMBX system service (permanent mailbox).

The Digital-supplied security profiles for devices in the evaluated configuration authorize all access types for SYSTEM and OWNER only; ² except for mailboxes. The profile for mailboxes includes all access types for GROUP and WORLD as well.³

Device access mediation is performed by the $QIO system service. On the first read and write operations performed by a subject, access authorization is checked; if authorized, subsequent read and/or write accesses are not reauthorized.

OpenVMS/SEVMS supports three types of I/O to devices: physical, logical, and virtual. An untrusted subject may perform only virtual I/O, unless the volume on the device is mounted foreign (see page 148, “Volumes”). Devices may be sharable, such as disk drives, or non-sharable, such as terminals.

**Discretionary Access Types for Non-Shareable Devices:**

- **READ** allocate and assign channels
- **WRITE** allocate and assign channels

READ, WRITE, or CONTROL access is required to allocate and assign a channel for the device. Physical I/O to unshared devices requires PHY I/O privilege.

**Discretionary Access Types for Shareable Devices:**

- **READ** issue read requests
- **WRITE** issue write requests
- **PHYSICAL** issue physical I/O requests
- **LOGICAL** issue logical I/O requests

**Mandatory Access Types for Devices:**

OBSERVE access is required to perform any I/O operations that retrieve data from the device, ALTER access is required to perform any operations that transfer data to the device or change the device physically.

SEVMS supports maximum and minimum classification labels for disk, tape, and printer devices. For disk and tape devices, when a volume is mounted on the device, the classification of the volume is the effective classification of the device (see page 148, “Volumes”). Only files with labels within the minimum and maximum device/volume labels can be written to the device. For printer devices, only files labeled within the range can be printed on the device.

For interactive terminal devices, a login range may be specified by a security administrator. A login attempt that is outside of the range for the terminal will be rejected. If no terminal login range is specified, the device

---

² Disk and tape devices include READ access for GROUP; but since the owner is SYSTEM, the group is a SYSTEM group.

³ The $CREMBX system service is the only TCB interface to create mailboxes. It requires protection data as an argument in the system call.
classification is used. The effective classification of the terminal for the duration of the interactive session is the user's login classification. The use of a login range allows the terminal device to be a single-level device, while still providing for a range of login classifications. The use of login ranges instead of terminal device ranges is required in the evaluated configuration.

System mailboxes (MBA1: (job controller), MBA2: (operator communication), and MBA3: (audit server)) are exempt from mandatory access controls since they are used for internal TCB communication only. They are protected from access by untrusted subjects using OpenVMS/SEVMS discretionary access control mechanisms.

5.2.3 File System Objects

The protected file system objects are volumes, files, and directories. This section describes the access characteristics of these objects.

5.2.3.1 Volumes

OpenVMS/SEVMS supports both “foreign” and “file-structured” volumes. A foreign volume is one that does not contain a standard OpenVMS/SEVMS volume label or, if it does contain a standard volume label, has been mounted as “foreign.”

For file-structured volumes (disk and tape), the volume protection data can be specified when the volume is initialized using the INITIALIZE command. If protection data is not specified, the volume security profile is used as the discretionary protection data. The Digital-supplied security profile for volumes authorizes all access types for SYSTEM, OWNER, GROUP, and WORLD. The protection data is written into the volume header; however, a tape volume has no ACL stored on the volume. For both disks and tapes, the UIC-based discretionary protection data stored on the volume may be temporarily overridden when the volume is mounted by a subject with CONTROL access or VOLPRO privilege. The protection data on a disk volume may be changed by subject with CONTROL access; however, to change the protection data on a tape volume, the volume must be reinitialized. To mount a “file-structured” volume as “foreign,” a subject must have the VOLPRO privilege or have CONTROL access.

Discretionary Access Types for Foreign Volumes:

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>issue read requests</td>
</tr>
<tr>
<td>WRITE</td>
<td>issue write requests</td>
</tr>
<tr>
<td>PHYSICAL</td>
<td>issue physical I/O requests</td>
</tr>
<tr>
<td>LOGICAL</td>
<td>issue logical I/O requests</td>
</tr>
</tbody>
</table>

Discretionary Access Types for File-Structured Volumes:

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>issue read requests</td>
</tr>
<tr>
<td>WRITE</td>
<td>issue write requests</td>
</tr>
<tr>
<td>CREATE</td>
<td>create files on the volume (disk only)</td>
</tr>
<tr>
<td>DELETE</td>
<td>delete files from the volume (disk only)</td>
</tr>
</tbody>
</table>

4 GROUP and WORLD access on volumes reflect the basic function of volumes in OpenVMS/SEVMS. For disk volumes, the individual files and directories on disk volumes contain explicit protection data (see page 149, “Files”). Tape volumes are used primarily for import/export and users may have different UICs on different OpenVMS/SEVMS systems.
SYSTEM and OWNER user categories (see page 156, “UIC-Based DAC”) always have READ and WRITE access to tape volumes. DELETE access for a disk volume also requires READ access.

**Mandatory Access Types for Volumes:**

A subject must have OBSERVE access to issue read requests and ALTER access to issue write requests to the volume.

In SEVMS, the volume’s classification range must fall within the classification range of the device on which it is mounted. The volume’s minimum and maximum security labels are the effective device labels while the volume is physically mounted. The subject’s classification must be within the volume’s range to mount the volume. To mount a labeled “file-structured” volume as “foreign,” the subject must have BYPASS or both UPGRADE and DOWNGRADE privileges.

In order to mount an unlabeled tape volume in SEVMS, the subject must have the IMPORT privilege and must specify an explicit classification to be assigned to the volume. In addition, non-labeled tapes may only be mounted for read-only access, unless the subject has BYPASS or both UPGRADE and DOWNGRADE privileges. To initialize a non-labeled tape, a subject must have BYPASS or both UPGRADE and DOWNGRADE.

### 5.2.3.2 Files

A file may be accessed by specifying either a file identifier or a pathname containing directory references used to locate the file (see page 111, “File Management”). To access a file via a pathname, a subject must have read access to all of the containing directories.

There is a hierarchy of mechanisms used to establish the discretionary protection data for disk files. Security profiles are not used for either disk or tape files. When a disk file is created, a subject may explicitly specify both the UIC-based protection fields and an ACL as arguments in the system service call. Alternatively, an argument may be specified to indicate that the protection data should be copied from the previous version of the file. If no explicit argument is included, there are two defaults associated with directories that can be applied to all files created within the directory. One default is for UIC-based protection; the other is for ACL entries. Both defaults are specified as ACL entries for the directory (see page 157, “The Access Control List”). If no directory defaults have been specified, a per-process default (for UIC-based protection only) will be employed. Finally, if protection data for the file has not been specified using any of the above mechanisms, a system default (RMS_FILEPROT parameter) is used for the initial protection data. The Digital-supplied system default authorizes all access types for SYSTEM and OWNER, plus READ and EXECUTE for GROUP. For the evaluated configuration, the TFM provides a system default setting that does not include any access for the GROUP category.

The protection data is stored in the file headers on the disk or tape. For files on tape volumes, the discretionary protection data for the volume (READ and WRITE access only) is applied to the files as well; no discretionary protection data is stored for tape files, only SEVMS mandatory labels.

**Discretionary Access Types for Files:**

- **READ**  issue read requests
- **WRITE**  issue write requests
- **EXECUTE**  execute an image or DCL file
- **DELETE**  delete the file
To write data to a file, a subject must have both READ and WRITE access. To use debugging tools on an image, a subject must have READ access to the image file.

**Mandatory Access Types for Files:**

A file cannot have a range of classifications; its label is set to the classification of the creating subject. A subject must have OBSERVE access to perform any operation on a file. A subject must have ALTER access to perform write or delete operations on a file. To delete a directory entry for a disk file, a subject must have ALTER access and discretionary WRITE access to the directory containing the file (see below).

While tape files do not have discretionary protection data, they do have a classification label. The label must be within the range specified for the volume. For both disk and tape files, a single classification label is stored in the file header of the file (see page 111, “File Management”).

### 5.2.3.3 Directories

Directories are disk files that contain the names and file identifiers of all of the disk files within the directory. Default discretionary protection data can be associated with each directory. When files are created, the default is used if no other protection data is explicitly specified. When a subdirectory is created, the default protection data associated with the parent directory is automatically copied to the subdirectory.

**Discretionary Access Types for Directories:**

- **READ** – read the entire contents of the directory
- **WRITE** – change the contents of the directory
- **EXECUTE** – read individual file entries (identified by name)
- **DELETE** – delete the directory

**Mandatory Access Types for Directories:**

A subject must have OBSERVE access to read the contents of a directory and both OBSERVE and ALTER access to create or delete a file entry within the directory.

### 5.2.4 Memory-resident Objects

OpenVMS/SEVMS support several in-memory objects that can be shared among subjects: global sections, common event flag clusters, logical name tables and resource domains. This section describes each of these objects and its associated security characteristics.

#### 5.2.4.1 Global Sections

A global section is a section of memory that can be shared among several processes (see page 89, “Global Sections”). A global section may be a disk-file section (paged to a disk file), page-file section (paged to a page file), or a page-frame section (physical pages). To create a page-frame section, a subject must have PFNMAP privilege. There are two namespaces for global sections: group and system. Group global sections are accessible only to subjects with the same GID; system global sections are accessible to all subjects. To create a system global section, the SYSGLBL privilege is required.
The process creating the global section explicitly specifies the UIC-based protection fields for a page-file or page-frame section. A specification of zero as the system call argument indicates that the security profile for global sections should be used as the protection data for the global section. The Digital-supplied security profile authorizes all access types for SYSTEM, OWNER, GROUP, and WORLD.

If the global section is a disk-file section, then the disk file protection data is used for the global section. Any changes to the protection data of the disk file will be reflected to the global section protection data; it is not possible to change the protection data of a disk-file global section directly.

A hardware access mode for the global section may be specified as an argument to the system service when the global section is created; the subject’s access mode is used as a maximum.

An untrusted subject may map any system global section into its virtual memory, subject to access mediation based on the global section’s protection data. Group global sections are accessible only to subjects in the same group.

**Discretionary Access Types for Global Sections:**

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>Read-only access to the pages in the global section</td>
</tr>
<tr>
<td>WRITE</td>
<td>Read/Write access to the pages in the global section</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>Map section in a protected hardware mode, for execution in a less protected mode (usable only from executive or kernel mode)</td>
</tr>
</tbody>
</table>

**Mandatory Access for Global Sections:**

A subject must have OBSERVE access to read the pages in a global section and OBSERVE and ALTER access to write to the pages of a global section.

### 5.2.4.2 Common Event Flag Clusters

A common event flag cluster is a mechanism for several processes in the same GROUP to share a set of event flags to synchronize processing (see page 104, “Common Event Flag Clusters”). To create a permanent common event flag cluster, the subject must have the PRMCEB privilege.

When a common event flag cluster is created, the ORB content is derived from the security profile for common event flag cluster objects, unless an argument explicitly indicates OWNER access only. The Digital-supplied security profile authorizes all access types for SYSTEM and OWNER, plus ASSOCIATE access for GROUP. The owner UIC is initially set to the UIC of the creator, and cannot be changed, since the name of the event flag cluster implicitly includes the GID of the owner.

**Discretionary Access Types for Common Event Flag Clusters:**

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSOCIATE</td>
<td>Associate with a common event flag cluster to set and test event flags in the cluster</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete a permanent cluster</td>
</tr>
</tbody>
</table>

**Mandatory Access Types for Common Event Flag Clusters:**

A subject must have both OBSERVE and ALTER access to associate with or delete a common event flag cluster.
5.2.4.3 Shared Logical Name Tables

Logical names provide a mechanism for substituting character strings for reference to devices, files, parameters, etc. (see page 105, “Logical Names”). Logical names reside in logical name tables that are searched when logical names are translated.

To create a shared logical name table, a subject must have the SYSPRV privilege. When a shared logical name table is created using the $CRELNT system service, a UIC-based protection field argument can be specified. If no argument is provided, the logical name table security profile is used. The Digital-supplied security profile for shared logical name tables authorizes all access types for SYSTEM and OWNER, and READ access for GROUP and WORLD.

Discretionary Access Types for Shared Logical Name Tables:

- **READ** Look up and translate names in the table
- **WRITE** Create and delete names in the table
- **DELETE** Delete a logical name table
- **CREATE** Create a descendant logical name table.

Mandatory Access Types for Shared Logical Name Tables:

A subject must have OBSERVE access to look up and translate names in the table. A subject must have ALTER access to delete a table. A subject must have OBSERVE and ALTER access to create and delete names in the table.

The system directory table (see page 105, “Logical Names”) and the (sharable) system logical name table are created during system initialization and are exempt from mandatory access controls. Modifications to either table can only be made by a subject with the SYSNAM privilege; all subjects may read logical names from both tables.

5.2.4.4 Resource Domains

A resource domain contains system resource names (locks) that share a common ORB (refer to page 106, “Resource Locks”). The $SETRESOURCE_DOMAIN system service establishes the subject’s authorized access to the resource domain; subsequent lock management system services base access decisions on the authorized access.

A resource domain is created by the OpenVMS/SEVMS TCB on first reference, and is never deleted. Two resource domains are always associated with a process: the system resource domain and a domain associated with the subject’s GID. The names of resource domains correspond to GID values. The resource domain security profile is always used as the protection data for a newly created resource domain. The Digital-supplied security profile authorizes all access types for SYSTEM, OWNER, and GROUP.

Discretionary Access Types for Resource Domains:

- **READ** Read lock value blocks
- **WRITE** Write lock value blocks
- **LOCK** Take and release locks ($ENQ,$DEQ).

---

5 The TCB creates JOB logical name tables during process creation, and automatically creates a GROUP logical name table for the user's GID if none exists at process creation.
Mandatory Access for Resource Domains:

The mandatory access controls of SEVMS only protect lock value blocks, and are not enforced on operations that enqueue and dequeue resource locks. A subject may take and release locks in any resource domain to which it has discretionary LOCK access. However, mandatory OBSERVE access is required to READ lock value blocks, and mandatory ALTER access is required to WRITE lock value blocks.

5.2.5 Queue Objects

OpenVMS/SEVMS manages queues to support the submission and execution of batch and print jobs. In general, the queues are managed by subjects with the OPER privilege; however, access to queues can be authorized for untrusted subjects to place jobs into the queue and to inquire about the status of jobs in the queue.

Discretionary Access Types for Queues:

- **READ** Query entries in the queue
- **SUBMIT** Place entries into the queue
- **DELETE** Modify or delete entries in the queue
- **MANAGE** Allows a subject to affect the contents of the queue

Each entry in a queue has an associated owner UIC, the UIC of the subject that placed the entry in the queue. Discretionary access mediation for entries in the queue is based on the owner UIC of the entry, as well as the protection data of the queue.

A subject with OPER privilege has MANAGE and CONTROL privilege to all queues, and may create and delete queues. When a queue is created, the queue security profile is used for initial protection data. The Digital-supplied security profile for queues is SYSTEM MANAGE, OWNER DELETE, GROUP READ, and WORLD SUBMIT.

Mandatory Access Types for Queues:

For SEVMS, a queue has associated maximum and minimum classification labels; and an entry has an associated classification label, equal to the label of the subject that created the entry. A subject must have OBSERVE access to both the queue and the entry to read entries in the queue, ALTER access to the queue to write entries to a queue, and OBSERVE and ALTER to perform any other operations on the queue.
Chapter 6

TCB Protection Mechanisms

This chapter describes the Trusted Computing Base (TCB) mechanisms that enforce the OpenVMS/Security Enhanced VMS (SEVMS) security policy and provide accountability. First, the Mandatory Access Control (MAC) and Discretionary Access Control (DAC) enforcement mechanisms are described. This is followed by detailed descriptions of the OpenVMS/SEVMS auditing, identification and authentication, and object reuse mechanisms. A final section on assurances discusses Digital’s hardware and software testing approaches, the OpenVMS/SEVMS informal security policy model, and Digital’s Rating Maintenance Phase (RAMP) plan.

6.1 Access Mediation

The OpenVMS/SEVMS TCB mediates access between the subjects and objects identified in Chapter 5, “TCB Protected Resources”. For untrusted subjects, it implements both a DAC policy and, in SEVMS, a MAC policy. Subject types and object types are identical for both DAC and MAC policies. Both policies must be satisfied for access to be granted.

In SEVMS, the MAC policy includes both a secrecy component and an integrity component. However, for the evaluation, the integrity component is not included in the policy; the use of the integrity component has no effect on the mandatory or discretionary policy of the TCB.

Although possession of any security-relevant privilege makes a subject trusted, certain of these privileges have a direct and explicit override effect on policy implementation. Privileges having a direct effect are discussed on page 159, “Access Mediation Privileges”.

Within a cluster, access to objects that are visible cluster-wide (volumes that are mounted cluster-wide, resource domains, and queues) is mediated on the subject’s node. To access a file on a non-cluster-visible disk that is on a different node, a subject may specify a DECnet node name as part of the file name. In this case, access is determined at the node where the file is located, based on the DECnet account used at the remote node (see page 174, “Authentication for DECnet Applications”).

6.1.1 Discretionary Access Control

The OpenVMS Discretionary Security Policy ensures that an object is accessible if and only if the subject is authorized for the access type required to perform the specific operation. Authorization is determined based on the previous actions by a subject with CONTROL access to the object. There are two distinct components of the discretionary policy. One component is based on access control lists and the other is based on categories of users (SYSTEM, OWNER, GROUP, WORLD) defined by the relationship of the subject’s User Identification Code (UIC) to the UIC of the object’s owner. This section first discusses the UIC-based mechanism and then describes the Access Control List (ACL) mechanism.
6.1.1.1 UIC-Based DAC

UIC-based discretionary access control defines four categories of users based on a comparison of the object’s owner UIC and the subject’s UIC. The four categories are SYSTEM, OWNER, GROUP, and WORLD. In the Object Rights Block (ORB) there is a protection field for each user category. The content of the field specifies the object access types that are authorized for that user category.

The User Identification Code (UIC) is an internalized representation of an individual user. It consists of a 16-bit User Identifier (UID) and a 14-bit Group Identifier (GID). A UIC is present in both the Access Rights Block (ARB) (the subject’s UIC) and the ORB (the object’s owner UIC).

A subject with GID less than or equal to the MAXSYSGROUP SYSGEN parameter is part of the SYSTEM category of users. A subject with the same UIC as the owner UIC of an object is in the OWNER category. A subject with the same GID as the owner UIC of an object is in the GROUP category. Any subject that does not fall into the SYSTEM, OWNER, or GROUP category is in the WORLD category.

A subject with the SYSPRV privilege is always in the SYSTEM category of users. Similarly, a subject with the GRPPRV privilege and the same GID as the owner UIC of an object is in the SYSTEM category for that object. Also, a subject with the same UIC as the owner of a disk volume is in the SYSTEM category for all files on the volume. If an object is owned by an identifier, rather than a UIC, then only the SYSTEM and WORLD categories are used for UIC-based access mediation.

6.1.1.2 Access Control List DAC

Access is granted or denied to arbitrarily defined groups or individuals using access control lists. An ACL for an object contains an ordered list of Access Control Entry (ACE) elements, some of which are identifier ACEs that specify object access types and a list of identifiers. Any subject holding all of the identifiers in an ACE is authorized for the associated access to the object. The first ACE of the list that matches the identifiers a subject holds determines the subject’s access. The matching ACE will either grant access regardless of UIC-based access, or deny access unless the subject is in the OWNER or SYSTEM categories.

Identifiers OpenVMS/SEVMS uses a concept of identifier to support access mediation using access control lists. A user can be authorized for a set of identifiers by a system administrator. The user’s UIC is itself an identifier, and is the index key for the other identifiers for which the user is authorized. The identifiers held by a subject are stored in Rights Lists within the ARB data structure. Each ACE within an access control list for an object also contains identifiers and an indication of the discretionary access types authorized for the identifier.

An identifier is represented internally as a 32-bit value, and externally as a string of up to 31 characters. The identifier may be either a 28-bit General Identifier or a 30-bit UIC. A General Identifier may represent (1) the user name associated with a UIC, (2) the group name associated with a GID, (3) an arbitrary name for grouping users, or (4) a system-defined identifier (BATCH, NETWORK, INTERACTIVE, LOCAL, DIALUP, and REMOTE).

When a user is granted an identifier, an associated identifier attribute may be specified. The attributes associated with an identifier are as follows:
DYNAMIC  Authors a subject to dynamically add or remove the identifier from the subject's Rights List.

RESOURCE  Authorizes the subject to create disk files that are "owned" by the identifier.

NO ACCESS  Specifies that the identifier is not to be used for access mediation decisions; it is normally used in conjunction with the RESOURCE attribute.

SUBSYSTEM  Authorizes the subject to specify a SUBSYSTEM ACE (see page 158, "Subsystem ACE") that refers to the identifier.

NAME HIDDEN  Limits the visibility of the identifier name to subjects holding the identifier and subjects authorized for READ access to the Rightslist database.

HOLDER HIDDEN  Limits access to the list of users who are holders of the identifier to only subjects who hold the identifier or have READ access to the Rightslist database.

**Rights List**  All identifiers are stored in the Rightslist database file, RIGHTSLIST.DAT. System services $FIND_HOLDER and $FIND_HELD support navigation from an identifier to the UICs holding it and from a UIC to the identifiers held by it. Every UIC holds at least one identifier—the one corresponding to that UIC. The $GETJPI system service can be used to determine the identifiers currently held by a process.

A process operates with a collection of identifiers in its Rights List. This collection is initially established as the union of identifiers attributed to the process and identifiers attributable to the cluster node. Typically, the only cluster node identifier is SYS$NODE, {nodename}, which identifies the node. The $GRANTID and $REVOKEID system services can add/delete identifiers, but require the CMKRL privilege. The SET RIGHTS_LIST command can be used by an untrusted subject to interactively enable or disable an identifier, if the subject holds the identifier with the DYNAMIC attribute.

**The Access Control List**  An ACL is a linked list of ACL segments, each containing one or more Access Control Entries. The list is pointed to by the ORB (and, for disk files and volumes, is also stored on disk). An ACL is established or modified by the $SET_SECURITY system service; an ACL editor is provided as a command level interface to $SET_SECURITY to manage the content of an ACL.

An Access Control Entry may be flagged with the following attributes:

**HIDDEN**  Indicates that the ACE is not visible using the ACL editor, except by a subject with the SECURITY privilege. (However, the ACE can be manipulated directly by an authorized subject via the $SET_SECURITY system service rather than via the command interfaces).

**PROTECTED**  Indicates that the entry cannot be deleted by global mechanisms (e.g., deleting the all of the entries in the ACL).

**NOPROPAGATE**  Indicates that the ACE will not be copied to later versions of a file under normal conditions.

**DEFAULT**  Indicates that the ACE is to be propagated to all files created within the directory; the propagated ACE does not retain the DEFAULT attribute except for subdirectories.

There are six types of ACEs used for security purposes: Alarm, Audit, Default Protection, Identifier, Creator, and Subsystem. Other types of ACEs are provided by OpenVMS/SEVMS, but they have no impact on security.
**Alarm ACE**  An alarm ACE can be used to specify that an alarm is to be sent to a security terminal when particular types of access are made to the object. (See page 166, “ACL Auditing”.)

**Audit ACE**  An audit ACE can be used to specify that an audit record is generated when particular types of access are made to the object. (See page 166, “ACL Auditing”.)

**Default Protection ACE**  A default protection ACE is only valid for a directory file object; but, it does not participate in the decision to grant access to that directory. It is used to specify the UIC-based protection fields to be applied by default when new files are created within the directory.

**Identifier ACE**  The essential parts of an Identifier ACE are the access type specification field and the identifier list. The access specification field is a mask whose bits specify yes or no for the granting of a request for access. The access types represented by a specific bit vary on a per-object basis as described on page 143, “Objects”. The most common meanings are CONTROL, READ, WRITE, EXECUTE, and DELETE. The identifier list is an array of identifiers (in internal form). If no bits are set in the access type specification, the identifiers in the list are not authorized for any access to the object.

**Creator ACE**  If a directory is owned by a general identifier, rather than a UIC, a Creator ACE may be used to specify an ACE to be included in all files created in the directory. The ACE explicitly defines the access authorized for the subject creating the file.

**Subsystem ACE**  A Subsystem ACE is valid only on an image file. If a subject holds an identifier with the SUBSYSTEM attribute, then that subject is authorized to create a SUBSYSTEM ACE having that identifier. The image file will then be part of a “protected subsystem” defined by the subsystem identifier. When the protected subsystem image is activated, the identifier within the subsystem ACE is added to the subject’s Rights List identifiers. This same identifier will have been specified as an identifier ACE on objects to be managed by the protected subsystem (e.g., its data files). While the protected subsystem image is activated, the process can have additional access inherited from the protected subsystem. The identifier is removed when the subsystem image is run down, the user interrupts the execution, or the user creates a subprocess. The general effect is that access to these objects is under the control of the protected subsystem.

### 6.1.2 Mandatory Access Control

The Mandatory Access Control (MAC) policy, applicable only to SEVMS, has three parts.

1. The **Simple Security Property** ensures that the clearance of the subject dominates the classification of the object before observation of the object takes place. (A subject cannot observe an object that has a higher classification label.)

2. The **Star Property** ensures that the clearance of the subject is dominated by the classification of the object before alteration of the object takes place. (A subject cannot alter an object that has a lower classification label.)
3. The *Tranquility Principle* prohibits the alteration of the assigned classification label of an object when
the object is actively accessed for observation or alteration. (The tranquility principle does not apply to
the enforcement of DAC; existing access to an object is unaffected by alterations of its DAC properties.)

The MAC access determination requires reference to the Simple Security Property when OBSERVE access is
requested, and reference to the Star Property when ALTER access is requested. Since some SEVMS objects
can have a range of labels, OBSERVE access compares the subject’s label to the object’s minimum label and
ALTER access compares the subject’s label to the object’s maximum label. In the evaluated configuration,
only disk and tape devices, spooled printers, volumes, and queues can have a label range. All of these objects
contain other labeled objects (e.g., volumes contain files, queues contain jobs). The MAC access check for
ranges allows any subject executing within the range to OBSERVE and ALTER the objects contained within
the ranged object. The label on the contained object determines the subject’s actual access to data.

6.1.2.1 Labels

SEVMS implements MAC labels on objects as a pair of classification blocks (maximum and minimum)
within the ORB; and it implements MAC labels on subjects as a classification block within the ARB. Each
classification block contains a secrecy level and secrecy categories. Specifically, each classification block
contains an eight-bit secrecy level and an eight-bit integrity level (interpreted as integers 0 to 255). SEVMS
provides two formats for a classification block, based on the second byte of the classification block. If the
block is of type 0, it contains 64 secrecy category bits and 64 integrity category bits. If it is of type 1, it
contains 128 secrecy category bits and no integrity category bits. The format used to label objects depends
upon the specific classification label required; type 1 format is used only for labels requiring secrecy categories
numbered greater than 64.

Dominance of one classification block over a second is determined when both (1) the numeric equivalent of
the secrecy level of the first is at least as great as the numeric equivalent of the secrecy level of the second,
and 2) the set of secrecy categories of the second is a subset of the set of secrecy categories of the first.
To effect the dominance test, either type of classification block is expanded to a virtual classification block
having 128 secrecy categories and 64 integrity categories by zeroing the unused categories.

An administrator can use the AUTHORIZE utility to associate a character string (up to 31 characters) with
each hierarchical level and each category.

6.1.3 Access Mediation Privileges

The privileges allowing immediate straightforward access, independent of object type are BYPASS, READALL,
DOWNGRADE, and, for the integrity policy, UPGRADE. BYPASS privilege allows unconditional access to
any object. DOWNGRADE privilege allows ALTER access to an object whose maximum classification does
not dominate the classification of the subject. READALL allows READ/OBSERVE access to any object.

In addition to these basic privileges, there are two privileges that allow a subject access as a SYSTEM
category subject (see page 156, “1UC-Based DAC”). A subject with the GRPRIV privilege is in the SYSTEM
user category for any object that has an owner GID equal to the subject’s GID. Similarly, a subject with
the SYSPRIV privilege is in the SYSTEM user category, independent of the subject’s GID. A subject in the
SYSTEM user category automatically has CONTROL access to an object.
Other privileges (such as OPER, VOLPRO, SECURITY, and IMPORT) affect access mediation to a specific class of objects. Each class of object may have an access exception routine that performs privilege checking for object-specific privileges.

6.1.4 Access Decision Function

The CHKPRO function performs the access decision algorithm within OpenVMS/SEVMS. Its decision is used by other components of the OpenVMS/SEVMS TCB to perform the actual access mediation. CHKPRO has both an external interface (the Check Protection (\$CHKPRO) system service) and an internal interface accessible only to the OpenVMS/SEVMS TCB. The internal interface is the central protection checking mechanism in OpenVMS/SEVMS.

The internal CHKPRO entry point has three input arguments that provide the data upon which the access decision is based: an ARB (see page 136, “Subjects”), an ORB (see page 144, “Object Rights Block”), and a data structure containing the requested DAC and MAC access types.

The request data structure contains the following fields:

- Requested Access Mode: Kernel, Executive, Supervisor, or User
- DAC Request Flags: Bit mask representing the type of access desired
- MAC Request Flags: OBSERVE and/or ALTER requested
- READALL Flag: Willing to use READALL privilege
- Audit Flags: Requested auditing options

The CHKPRO function determines whether a subject with the specified rights and privileges is authorized to access an object with the specified attributes. The CHKPRO function does not itself grant or deny access; rather, it is available to any TCB component that is making an access control decision. The system services \$CHKPRO and \$CHECK_ACCESS allow untrusted subjects to invoke the CHKPRO function. \$CHECK_ACCESS uses a symbolic object name and a symbolic user name as input while \$CHKPRO uses an abstract description (an item list) to describe the object and subject. In either case, a subject may only check access to objects it is already authorized to access.

As a matter of Digital policy, the CHKPRO function is always used by the TCB to perform access mediation.

The CHKPRO function returns the access decision, together with the ACE that allowed or denied access (if appropriate) and a mask indicating any privileges required for a grant access decision. CHKPRO optionally calls the \$AUDIT_EVENT system service to audit the access decision (see page 163, “Auditing”).

The normal user needs some understanding of how the CHKPRO function makes the access decision, since this decision is closely related to policy and affects the protection of owned objects and the accessibility of objects used. The Guide to VAX/VMS System Security [34] discusses the steps in the decision, except for the mandatory access control aspects that are described in the VMS SES User’s Guide [48].

6.1.4.1 Access Decision Algorithm

This section describes the algorithm used by the CHKPRO routine to grant or deny access.
Processor's Access Mode Check  The ORB contains a hardware access mode field; this first access mediation check compares the subject’s access mode with the mode protection vector in the ORB. However, in the evaluated configuration, this check is skipped, since OpenVMS/SEVMS always creates objects without a valid access mode (i.e., the mode_valid bit in the ORB is clear) and there is no TCB interface to set an object’s access mode. Therefore, this check never denies access.

MAC Check  If the CLASS_PROT SYSGEN parameter is set (e.g., for SEVMS), then the MAC check is performed; otherwise this step is skipped.

If MAC OBSERVE access is requested, the secrecy label of the subject must dominate the minimum secrecy label of the object (and the integrity label of the subject must be dominated by the maximum integrity label of the object).

If MAC ALTER access is requested, a check is made that either (1) the secrecy label of the subject is dominated by the maximum label of the object, or (2) the subject had the DOWNGRADE privilege. It is also checked that either (1) the integrity label of the subject dominates the minimum integrity label of the object, or (2) the subject has the UPGRADE privilege.

If any MAC check fails, then transfer is made to check if the subject has the BYPASS or READALL privilege (page 161, “BYPASS/READALL Privilege Checks”).

DAC Check Using Access Control List  Each Identifier ACE in the ACL is checked for a match on the list of identifiers. If a match is found and that ACE grants access, then access is granted and transfer is made to the object-specific check. If a match is found and that ACE denies access, then, in the following UIC check, only SYSTEM and OWNER categories will be considered.

If for any reason the ACL is determined to be invalid (e.g., due to data corruption), transfer is made to the BYPASS and READALL privilege checks.

DAC Check Using UIC  Checks for OWNER access are made first, since it will be the most common. If the process is the owner (its UIC matches the object’s UIC) and the owner protection field in the ORB allows the access types requested, then access is granted and transfer is made to the object-specific check.

Otherwise, checks are made for WORLD access. If the world protection field allows the access type requested, access is granted and transfer is made to the object-specific check.

Otherwise, checks are made for GROUP access. If the group protection field allows the requested type(s) of access, and the subject’s GID matches the group of the object’s owner, access is granted and transfer is made to the object-specific check.

Otherwise, checks are made for SYSTEM access. If the subject either has a system GID or has SYSPRV privilege, and the system protection field allows the requested access, then access is granted and transfer is made to the object-specific check. If the subject has GRPPRV and has a GID the same as the owner GID, access is granted and transfer is made to the object-specific check.

BYPASS/READALL Privilege Checks  At this point, access has not yet been granted; the BYPASS and READALL privileges are now checked to see if they would override a denial of access. If the subject has
BYPASS privilege and write (alter) access is requested, then access is granted and transfer is made to the object-specific checks.

If READ (OBSERVE) access has been requested and (1) write access has not also been requested, and (2) the subject has READALL privilege, and (3) the READALL Flag indicates that the privilege can be used, then access is granted and transfer is made to the object-specific check.

Finally, if the subject has BYPASS privilege, access is granted; otherwise access is denied and the object-specific checks are performed.

Object-Specific Check  At this point, access has either been granted or denied. An object-specific access exception routine is now called to determine, based on the type of object, if that decision should be changed.

Object classes that have access exception routines are queues, logical name tables, volumes, and common event flag clusters. These routines check object-specific system privileges (e.g., OPER for queues, VOLPRO for volumes). If the subject has the required privilege, access may be granted, independent of the access decision that had been made.

Audit  Several system flags are checked to determine if CHKPRO should perform the audit, or if the TCB component calling CHKPRO will perform any necessary auditing. If auditing is selected, CHKPRO builds an audit record and calls $AUDIT.EVENT (see page 167, “Event Reporting”).

The final access decision is then returned to the caller.

6.1.5 Security Object Management

OpenVMS/SEVMS provides a set of routines that provide object-specific security functions. The standard OpenVMS/SEVMS security functions ($GET_SECURITY, $SET_SECURITY, and $CHKPRO) call the object-specific functions to perform operations that vary depending on the type of object.

Information about each class of object is maintained in the Object Class Block (OCB) for the object class. This information includes a definition of the access types for the object class, one or more default security profiles, default audit and alarm settings, and the addresses of the object-specific functions supported for the object class. The OCB is created when the object class is registered during system boot. When objects are created, the address of the associated OCB is stored within the ORB.

Each security object manager provides a set of object-specific functions that include an access exception routine used by $CHKPRO (see page 160, “Access Decision Function”), a routine to retrieve object class information (e.g., character string access type names), routines to copy and update the ORB for an object, and routines to initialize and terminate access to objects when viewing and changing the object’s security attributes.
6.2 Auditing

The audit functions are included in system services, Digital Command Language (DCL) commands, the audit server, and audit analysis services. The basic audit design is the same for OpenVMS and SEVMS. However, for SEVMS, additional events are auditable and the associated classification labels are included in the audit record.

The following is a list of the auditable events (OpenVMS and SEVMS):

- Use of the SET AUDIT command
- Changes to the authorization databases (SYSAF, RIGHTSLIST, and NETPROXY)
- Use of the INSTALL utility
- Use of the SYSGEN utility
- Use of identification and authorization (login, logout, login failures, and break-in attempts)
- System time modifications
- Object creation, access (including setting and changing security attributes), deaccess, and deletion
- Use of process control services
- Volume mounts and dismounts
- Use of privileges
- Use of identifiers as privileges
- Network management changes
- DECnet connection requests, acceptances, rejections, and disconnections.

Figure 6.1 depicts the basic architecture and data flow of the audit design. Each event can trigger an alarm message, an audit message, or both. Alarm messages are sent to all SECURITY terminals and, optionally, to a listener mailbox; but they are not recorded on disk. Audit messages are eventually stored on disk in the audit log and are also sent to the listener mailbox, if one is enabled. Audit messages may also be copied to the archive file, if the event type meets the archive criteria.

6.2.1 The Audit Server

The actual audit logging is performed by the audit server that runs on every node of a cluster. The audit server runs as a detached process (i.e., it is a root of a process tree) and is created during system startup. There is one audit log file that resides by default on the system volume (on the boot node of a cluster). Coordination of input into the log file is performed by lock management. There may also be an associated security archive file (a backup for the log file), and a listener mailbox. The audit server writes audit records to all three as each event is processed.

The audit server runs with the following privileges during normal processing: ALTPRI, CMKRN, DETACH, NETMBX, OPER, PSWAPM, SETPRV, SYSLCK, SYSPRV, TMPMBX, WORLD. All privileges are enabled during server error processing (this occurs inside the condition handler). The SETPRV privilege is used only in the condition handler to ensure that a dump file can be written and that the server can create and activate a new copy of itself if necessary.

All audit records to be logged arrive in the form of messages into the audit server’s mailbox; that is a system mailbox created for use only by the auditing subsystem. There is no direct programming interface to the audit server. The server’s mailbox is owned by the SYSTEM account and is protected by denying access
CHAPTER 6. TCB PROTECTION MECHANISMS

Figure 6.1. Functional Flow of Audit Mechanism
to all other categories of users. All writes to the mailbox are in kernel mode via an internal entry point in the mailbox device driver that is called from an internal entry point in the Audit Event ($AUDIT_EVENT) system service. The audit server is awakened to read messages from the server’s mailbox by the delivery of an Asynchronous System Trap (AST) from the mailbox device driver. As the server reads messages, they are put into one of two internal work queues depending on the type of message: control or event. All control messages are processed before the event messages are handled.

During audit event processing, the server will work on all of the messages in the internal work queues and then suspend. If the message contains an audit and alarm record, the message is forwarded to the operator terminals (terminals designated as SECURITY terminals). Event messages are recorded in the audit log. The audit log is protected by restricting DAC access to the system user category and privileged processes. Errors are recorded in a separate audit server error log.

A listener mailbox may be created to receive a binary copy of each incoming audit event. The intended use of this capability is for an artificial intelligence tool to receive audit messages in real time and to process them. The listener mailbox has the protection specified by the subject who creates it. The listener mailbox must be specified with a name and must exist prior to the issuance of the “SET AUDIT/LISTENER=” command.

The audit server also manages the audit event bitmasks that indicate the events (of the items listed on 163) to be logged. The bitmasks are set via the DCL command SET AUDIT. The audit server also monitors the disk space on the system volume to be sure there is enough space for the audit log file. The server cannot be deleted, suspended, or have its priority lowered. It is started by default, with the exception of a minimum boot used during resource exhaustion procedures.

The parameters by which the audit server is governed are contained in a database specifically set up for the server (one per cluster). The database contains the alarm and audit event bitmasks, failure mode settings, resource exhaustion mode flag, resource exhaustion thresholds, final action setting, archive class and destination, and audit log destination. The database information is altered by the server in response to control messages.

### 6.2.2 Audit Controls

There are three different methods used to control the amount of audit information generated. The first method is the DCL commands SET AUDIT, SHOW AUDIT, and REPLY. The second method is the use of ACL entries. The third method is user auditing, performed by settings in the User Authorization File (UAF). These methods are described in the paragraphs below.

#### 6.2.2.1 DCL Commands

The SET AUDIT command defines the parameters in the audit server database and also directs the server to perform certain audit-related functions. The command sends a control message to the server to cause the server to write information into the database, if necessary, or to perform the specified function. The SECURITY privilege is required to invoke SET AUDIT. BYPASS privilege is used in command processing to write a message to the server mailbox. Once the the message is written, the BYPASS privilege is disabled. Usage of the SET AUDIT command is itself audited. The following are the types of control messages processed by the server:
Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha

CHAPTER 6. TCB PROTECTION MECHANISMS

- Enable/disable audit and alarm events
- Add/remove audit journals
- Flush a journal; flush all journals
- Enable/disable archiving
- Modify archive destination or class
- Enable/disable listener device
- Modify server internal timer intervals
- Set bad journal cache limit
- Shutdown or restart server
- Create new cluster-wide audit log
- Create node-specific log (not in evaluated configuration)
- Redirect audit log
- Modify alarm or failure mode settings
- Modify resource exhaustion mode settings
- Lock server against control functions
- Modify the action, warning, and resume thresholds
- Request an immediate resume scan
- Add/remove a process from the resource exhaustion process exclusion list
- Perform system snapshot processing.

SHOW AUDIT is a DCL command that lists the events for which auditing is currently enabled. The SECURITY privilege is required for running SHOW AUDIT.

The REPLY/ENABLE and REPLY/DISABLE commands are used to enable and disable an operator’s terminal to receive (or not receive) alarm messages. Alarms are described below.

6.2.2.2 ACL Auditing

Through the use of Audit and Alarm ACEs, both security alarm and audit messages can be generated for accesses to specific OpenVMS/SEVMS objects. Alarms and audits can be set to trigger on any success or failure of any kind of object access attempt (e.g., CONTROL, READ, WRITE). A system-wide audit parameter must also be set to enable ACL auditing.

6.2.2.3 Auditing the Actions of a User

Through the UAF and its associated utility, AUTHORIZE, the actions of a particular user can be audited. If this feature is used, a bit in the Process Control Block (PCB) is set that enables auditing for all security-relevant actions initiated by processes activated by the user.

6.2.2.4 Auditing Operator and Administrator Actions

Since all use of privilege may be audited, the operator and administrator actions performed directly through the system can all be audited. In the case of the Hierarchical Storage Controller (HSC), a printer is attached to the operator console that is used to record all commands entered at the console. For the evaluated
configuration, manual procedures are required to associate operators with the actions recorded on the printer, as described in the OpenVMS/SEVMS Trusted Facility Manual (TFM).

6.2.3 Event Reporting

Audit event reporting is performed through four system services and several internal routines. The system services are $AUDIT_EVENT, Check Privilege ($CHECK_PRIVILEGE), $CHKPRO, and Check Access ($CHECK_ACCESS). $AUDIT_EVENT supports an external (used by most callers) and an internal (used by $CHKPRO) entry point, but provides the same auditing functions regardless of which entry point is used. $AUDIT_EVENT is the system service used to record all security-relevant events with the exception of most object accesses and use of privilege. The $AUDIT_EVENT service is used with file access auditing. The determination of whether to actually record the event occurs within $AUDIT_EVENT as well. The auditing decision is made based on the events that are enabled by the system administrator.

The $AUDIT_EVENT service takes the caller’s input and determines whether the event needs be audited. If so, it writes the audit record to the audit server’s mailbox. The basic audit record contains the following information: event, time, process ID and name, username, process owner UID, account name, and classification. The content of the rest of the audit record varies depending on the event being audited. The caller can optionally request verification of the audit by specifying one of two parameters to the $AUDIT_EVENT service. If the AUDITS parameter is specified, a temporary mailbox is created that is used to receive information from the server. In that case, the service may wait for a reply from the server before returning to the caller. The temporary mailbox is then deleted. The other option involves use of an AST via the ASTADR parameter. Calls to $AUDIT_EVENT from the TCB do not request confirmation from the server.

$CHECK_PRIVILEGE is used by the TCB to audit the use of privileges and certain identifiers. The service determines whether the caller possesses the privileges needed to perform a particular function. For the specified privileges, if auditing is enabled for those privileges, $CHECK_PRIVILEGE records success or failure of use by sending the appropriate message to the audit server. $CHECK_PRIVILEGE performs the same function using identifiers instead of privileges; this allows identifiers to be employed as application privileges. The reason for the attempted use of privilege is included in the audit record by the caller.

The use of $CHECK_PRIVILEGE to generate an audit record requires either the AUDIT privilege or execution in kernel or executive modes. An unprivileged process may use $CHECK_PRIVILEGE to test for privileges without involving auditing. The mailbox mechanism described above is also used in processing. Internal check privilege routines perform support processing.

As an option to the $CHKPRO system service, a caller may request that the object access be audited. $CHKPRO uses an internal entry point to $AUDIT_EVENT to perform auditing. The same option is available in the $CHECK_ACCESS service. $CHECK_ACCESS calls an internal entry point in $CHKPRO that results in recording the event (via the same entry point mentioned previously). See page 160, “Access Decision Function”, for more details on $CHKPRO and $CHECK_ACCESS.

When acting on behalf of a subject, a TCB component may temporarily turn off auditing to avoid generating multiple copies of the same audit record or to avoid recursively generating audit information. A bit in the PCB is set to indicate that auditing is bypassed for the process. This is set by the TCB component itself. An example of this is the $AUDIT_EVENT system service that sets the bit off during event processing.
6.2.4 Error Processing

Audit error conditions that occur in process context result from resource exhaustion conditions. Such conditions are discussed below. Fatal error exceptions (those that occur inside the audit server) are discussed on page 169, “Audit Server Errors”.

6.2.4.1 Resource Exhaustion

Resource exhaustion issues were given much attention in the design of the audit server. The audit server monitors its needed disk resources in a manner designated by the system administrator. Resource monitoring is based on two thresholds: space and time. The space threshold is based on the number of free disk blocks on the volume or volume set. The time threshold ensures there is space on disk for a specified number of seconds worth of messages, assuming an average message size. The server will attempt to extend the audit log for the amount of space it needs to store records until it encounters an error. When the extension attempt fails, the server acts in accordance with threshold settings. There are three thresholds associated with the server and its resources: warning, action, and resume. The modes and thresholds may be dynamically altered by a system administrator.

When the free space on the system volume shrinks below a warning threshold specified in the server database, the server initiates resource exhaustion processing. Warning messages are sent to the SECURITY terminals once per minute as long as the free space remains below the threshold. If the free space diminishes below an action threshold, the server will suspend activity on the cluster to reduce the rate of audit information being generated. At that point, only operators can log into the system. System activity resumes after the free space grows above the resume threshold. The SET AUDIT/SERVER=RESUME command can be used by the system administrator to force the server to reexamine the free space while the server is in a resource exhaustion state.

The SET AUDIT command is also used to add, change, and remove audit logs and to enable or disable archiving. When space is at a premium, for example, a user with SECURITY privilege can reroute the server’s output to another file and send the log(s) and archive(s) to tape.

For memory allocation failures, the server will retry the request until it is satisfied. Any events being processed when the server is in a resource exhaustion state are buffered into an internal message cache in process virtual memory. The internal server cache is initially 512 Kbytes. The cache is expanded by half its size on cache write failures. The size of the cache (and of a message) is limited by the amount of process virtual memory. The rate at which the cache fills up is dependent on the number of events being audited and the number of auditable events that occur. The default interval after which the message cache is flushed and records are written to disk is five seconds. The flush interval may be altered by a system administrator.

6.2.4.2 Flow Control

To limit the possible loss of audit data, a flow control mechanism is incorporated into the audit subsystem design. The flow control mechanism operates independently of the resource monitoring modes described above. There are six thresholds associated with the internal event message queue (there is a different queue for control messages that is currently not subject to the flow control mechanism described herein). Three are for the queue itself; the other three are associated with processes for which audit data has been entered into the queue. The thresholds are treated in increasing order of severity. As the queue and process thresholds
are met or exceeded, processes are suspended by the audit server with the exception of those on a process exclusion list (e.g., SWAPPER, OPCOM, and NETACP). The defaults for the queue thresholds are 100, 200, and 300 messages, respectively. The default process thresholds are 5, 2, and 1 messages. The process thresholds indicate that a process may have no more than some number of messages in the queue, depending on which threshold is of concern. The thresholds may be altered by a system administrator through the SET AUDIT command, except for the third process default. If a process is suspended, then all messages in the queue for that process must be logged before the process is resumed. If the queue becomes full despite the flow control (i.e., processes on the exclusion list generate enough audit data to fill the queue) and the queue cannot be expanded, then processing proceeds according to the internal server error processing described below.

6.2.4.3 Audit Server Errors

Audit server errors occur as a result of resource exhaustion conditions, network problems, and other internal failures. There are three error conditions processed by the server that may affect the amount of audit data recorded by the system: failure to allocate an internal message buffer element (or to expand the buffer due to the failure to allocate more memory), failure to write to the audit log because of a lack of free disk space, and failure to write to the archive file because of any network error condition. There are four possible modes in which the server can process these errors: ignore new events, purge old events (those in the message buffer), crash, and restart. The default setting is purge.

In the purge mode, for message buffer allocation failures, the oldest message in the internal message buffer not yet written to the audit log is discarded in favor of the incoming message. In the ignore mode, all incoming messages are ignored until buffer space is available. In the case of a failure to write to the audit log, if the server error setting is either purge or ignore, the server reports a journal full message to the CENTRAL and SECURITY class terminals and terminates. A failure in crash mode results in a system shutdown. The restart mode exists for debugging and testing purposes. If the server is to be restarted, all messages are flushed from the internal queues and the audit log and archive file are closed. The server generates a process dump and creates another copy of itself. Any queued audit event messages are lost during the restart. The server may also be restarted using the SET AUDIT command. A system administrator can also control the server error processing mode via the SET AUDIT command.

If the server fills the buffer, an attempt is made to allocate more memory with which to expand the buffer. If the request cannot be filled, because the server's virtual memory limit was reached, then the server failure mode setting is used to determine the course of action. In the case where there is not enough free disk space to write to the audit log, if resource monitoring is disabled (by default, resource monitoring is enabled for the disk on which the audit log resides), the server will perform processing according to the server error mode setting as described above. The case in which resource monitoring is enabled is handled to the extent that it can be by the flow control mechanism. If the third-level flow control thresholds are entirely exceeded, then server error processing is invoked. Server errors are processed in the condition handler established during server initialization. In the case of network failures that occur in writing to the remote archive file, the server issues messages to the CENTRAL and SECURITY class terminals whenever a network link state transition is detected and once a minute if the archive file is not available.

For errors considered non-fatal, the server reports the error (by sending a message to all CENTRAL and SECURITY class terminals), ceases the current operation, and continues processing. Other errors considered
fatal are processed in the condition handler of the server. Support for restarting the server is contained in the condition handler. If the server is to be restarted, any audit messages that are written to the server mailbox during the restart are processed by the new copy of the server. Any messages in the internal server queues are lost. However, the restart processing marks each entry in the message queue to indicate to the system that as much of the queue as possible will be fit into the dump file when a dump is generated. Fatal error mode processing is only indirectly alterable by a system administrator through the setting of the server error processing mode.

### 6.2.4.4 Process Context Errors

Process context errors are those that occur in attempts to deliver an audit event to the server. The resource exhaustion condition for process context errors occurs when the server’s mailbox is full. Resource exhaustion processing is, for the most part, done transparently to the calling process. The calling process is put into a wait state until the needed resource is freed up. The process is then rescheduled. Assuming the process tries again to process the audit event, another attempt is made to write the message to the audit server’s mailbox.

### 6.2.4.5 Loss of Audit Data

Though significant loss of audit data is unlikely since the flow control mechanisms take precedence over resource exhaustion processing, loss of audit data may occur either when the system crashes or when the audit server aborts.\(^2\) If the system crashes, the contents of physical memory and virtual memory of all processes including the audit server are saved in a system dump. The new end-of-file mark for the audit log and archive file may be lost, which will raise the possibility of some records in the log and/or archive being inaccessible.

If only the server aborts, then the data in the server’s process address space and in the server’s mailbox is not written to the audit log. However, the end-of-file mark, if not written to disk, may be lost, too. The loss of records due to the end-of-file mark not having been written may be reduced in a cluster since other servers still active may have current information.

In the case of system crashes, if the rate at which audit records are generated is slower than the rate at which records can be written, the potential loss of audit data is \([\text{audit-rate}\times\text{flush-interval}]\). The flush interval is typically 5 minutes.\(^3\) If the audit rate is faster than the writing rate, the potential loss of audit data is then \([\text{writing-rate}\times\text{flush-interval}]\) plus the tertiary backlog plus the number of memory-resident processes. The worst case audit data loss occurs in the following scenario: the system crashes; every memory resident process (100 memory-resident processes is common) is prepared to send an audit record; and the tertiary queue threshold (300, as mentioned earlier) is exceeded. There may be several additional audit records lost from processes on the exclusion list, depending how quickly they are written to the queue before the hypothetical system crash occurs. Further, if the audit server is somehow blocked from execution prior to the system crash, then it is possible that many messages in the server’s mailbox (maximum 65,535 bytes of data) may be lost. To make the operator aware of audit data loss issues, the disk space monitoring and flow control design are discussed in the TFM.

---

\(^2\) There is a third case in which the server’s mailbox overflows. This case is considered a rarity. If the mailbox fills, by default, a system shutdown results. The data is not written to the audit log, but is contained in the system dump.

\(^3\) “Flushing” is essentially the process of updating the end of file mark in the file header of the audit log.
6.2.5 Analysis Tools

Audit analysis is supported through the Format an Audit Record ($FORMAT_AUDIT) system service and the ANALYZE/AUDIT utility. $FORMAT_AUDIT formats a binary message recorded in the audit log or listener mailbox. The service is used by the ANALYZE/AUDIT utility to create a human-readable audit record.

ANALYZE/AUDIT selects audit records from the audit log based on user-specified criteria. ANALYZE/AUDIT provides a flexible array of selection criteria. A user may select audit records based on various fields within the audit record, such as username, date, and event type, such as use of a particular privilege or failed login attempts. In addition, a range of dates may be specified to select audit records. A report can be generated with either a brief (one line per record) or full listing of each audit record selected. The output is written either to the screen or appended to a file. ANALYZE/AUDIT is a non-privileged command. However, the user must have appropriate discretionary access to the audit log.

Reconstruction of the sequence of events in a cluster is aided through synchronization of the clocks on each node. The TFM calls attention to the issue of clock synchronization and references a command procedure that can be repeatedly executed after a specified time interval to reduce drifting of cluster node clocks.

6.2.6 Startup and Shutdown

Some audit-related functions are performed during system startup and shutdown. The system alarm and audit event bitmaps are allocated and read in to non-paged pool. The audit server is activated during system startup if specified via the SET AUDIT command. The server then reads its database and restores its bitmaps, SEVMS classification audit setting, failure mode settings, and archiving and resource exhaustion states. The server then writes a startup event into the audit log. The system startup procedure then continues. During shutdown, auditing is disabled.

6.2.7 DECnet Auditing

All successfully parsed Network Control Program (NCP) commands issued to the DECnet software are audited. All of the command information, including completion status, is included in the audit record with the exception of passwords, which are filtered out. It is possible to audit both successful and unsuccessful creations and terminations of logical links for both incoming and outgoing connections. Additional DECnet information recorded in the audit record includes the following: logical link number, DECnet application name and application number, remote nodename, remote node identification, remote username, and remote logical link number. Use and attempted use of privileges for DECnet operations are also auditable.

6.3 Identification and Authentication

This section describes the identification and authentication mechanisms provided by OpenVMS/SEVMS. The features supported by OpenVMS/SEVMS to control access to the system, to limit the environment accessible to specific users, to provide a trusted path for login authentication, and to authenticate remote DECnet connections are described.
6.3.1 User Accounts

For a user to login to an OpenVMS/SEVMS system, the user must have an associated user account registered by a system administrator in the UAF. The account may be open, requiring no password; it may be a normal user account requiring a single password; it may require both a primary and a secondary password; or it may have a password that is controlled by a system administrator rather than the user. If an administrator has specified that a system password is required for a particular terminal device, then, before a user enters any account information, OpenVMS/SEVMS requires the entry of the system password. A terminal may be specified as an autologin terminal, where all logins from the particular terminal are automatically associated with a particular user account, without any password authentication. Open accounts and autologin terminals are specifically disallowed in an evaluated configuration as described in the applicable OpenVMS/SEVMS TFM.

6.3.2 Account Restrictions

OpenVMS/SEVMS provides a number of alternatives to constrain the scope of a user’s session. A user account can be made totally inaccessible without removing the account from the UAF database, either through direct action or via an account expiration date. A user account can have restrictions associated with it that limit BATCH, DIALUP, NETWORK, INTERACTIVE, LOCAL, and REMOTE access, as well as a specification of time-of-day ranges associated with each type of access.

An account may be set up as a captive account that constrains the user in the following ways:

- Disables access to DCL commands.
- Inhibits the use of user-generated interrupts (CTRL-Y, for example).
- Prevents the use of certain command qualifiers with the login command (/CLI, /COMMAND, etc.).
- Assures that the system and user login command procedures are executed.
- Prevents processes from being spawned from the VMS Mail and Editor commands.
- In SEVMS, prevents the user from specifying a classification at login.

An account may be set up as a restricted account, that has all of the same restrictions as a captive account except that access to DCL commands is not restricted following the login processing. An additional account restriction can prevent a user from running user-written programs.

When the SYSTEM account is used to login from the system console, any account restrictions specified in the UAF are not imposed. Since the SYSTEM account is an administrator account, the restrictions can only be bypassed by an administrator with physical access to the system console.

6.3.3 Passwords

Users may control their passwords to the extent determined by a security administrator. The restrictions available to an administrator for enforcing a password management policy include the following:

- Minimum password length
- Password expiration dates
• Automatic password generation, either required or optional
• Dictionary screening of proposed passwords
• History of previously used passwords.

The password management policy is specified for each account through the use of the AUTHORIZE utility. A subject with the SECURITY privilege may change the system password.

6.3.4 Login Processing

Login processing is discussed in detail in “Processes with a CLI,” on page 102. The identification and authentication functions are performed by the LOGINOUT image that is activated by the job controller when unsolicited input is received from a terminal device. LOGINOUT prompts for the username (account) and verifies that the account is valid. The LOGINOUT image prompts for all passwords without echoing the password, and encrypts the passwords using a one-way encryption algorithm before comparing them to the passwords stored in the UAF.

When the DECnet remote terminal application is employed to create a session on a remote node (using the SET HOST command), a pseudo-terminal is created at that node to handle terminal I/O. The LOGINOUT processing is identical.

6.3.5 Secure Terminal Server

OpenVMS/SEVMS provides a trusted path for login processing through the use of the secure terminal server. A security administrator may specify an option at system startup for each terminal device to require the use of the BREAK key to initiate a connection. Since a new password can be specified during login processing, the trusted path can also be used for changing passwords on those terminals that use the secure terminal server.

To implement secure terminals, the terminal device driver treats the BREAK signal, rather than the RETURN key, as the unsolicited terminal input that initiates an interactive login session. The secure terminal server is only available for directly connected terminal devices, and not for workstation display devices or remote DECnet sessions.

6.3.6 Break-in Controls

OpenVMS/SEVMS provides a mechanism to assist in the control of break-in attempts. By default, a user is allowed three username/password attempts, each separated by up to a 20-second interval before OpenVMS/SEVMS causes disconnection. When the count of login failures in a specified time period exceeds a site-specific limit, OpenVMS/SEVMS assumes that a break-in attempt is in progress. A system administrator can specify terminal ports and usernames to be tracked by the break-in facility. If the system determines that a break-in attempt is in progress, it initiates evasive action by prohibiting further login attempts from the suspected terminal and username. After a specified time period, logins will be permitted; however, OpenVMS/SEVMS provides an alternative where the suspected account is made inoperative until a system administrator re-enables it. Data concerning suspected intrusion can be captured not only in the audit trail (if login failures are enabled) but also in an intrusion database.
6.3.7 Authentication for DECnet Applications

For a DECnet connection (see page 85, “DECnet”), there must be a correspondence between the local subject on the source node and the remote subject on the target node. In SEVMS, DECnet assures that an unprivileged remote subject has the same classification as the local subject on the source node. This section describes how DECnet determines the user identity of remote subjects when a connection to a DECnet application is requested.

6.3.7.1 DECnet Authorization Options

There are four different options for determining the correspondence between a local subject and the remote subject: explicit access control strings, proxy accounts, DECnet application accounts, and the default DECnet account. The applicable OpenVMS/SEVMS TFM provides guidance and cautions about the use of these different mechanisms, and excludes default DECnet accounts from the evaluated configuration.

Explicit Access Control String In a DECnet request, the local user can specify a username/password combination for use on the target node. The authorization database on the target node must contain an account authorization that corresponds to the specified username and password. For example, a file specification may contain a DECnet node name and a username/password pair. DECnet will create a File Access Listener (FAL) process that has the account characteristics of the specified username and, as a result, will attempt to access the file using the authorizations of that user.

The applicable OpenVMS/SEVMS SFUG and TFM describes the use of explicit access control strings and provides advice and cautions about how they should and should not be used. Any passwords specified in explicit access control strings are communicated unencrypted over DECnet.

Proxy accounts A proxy account provides a mechanism for an account on a source DECnet node to be associated with a particular proxy account on a target DECnet node. When a connection to the target node is made, DECnet creates a process using the account characteristics of the proxy account. A system administrator on the target node establishes the valid associations between the source node username and the target node proxy account. In the evaluated configuration, since the source and target nodes share a common security database, user accounts and proxy associations (if any) will be identical on all nodes.

DECnet Application Accounts Each DECnet application may have an associated default account. All connections to the application would have the same target account characteristics. DECnet applications that are part of the TCB (e.g., Mail) generally have default application accounts.

Default DECnet Account This account allows all network applications general access to the DECnet node. There is no need to specify username and password. The OpenVMS/SEVMS TFM states that a default DECnet account is invalid in an evaluated configuration.
6.3.7.2 DECnet Security Administration

An administrator on a target node establishes the DECnet authentication policy. In the evaluated configuration, the DECnet authorization files are considered part of the security database; there will be a common policy on all nodes in the cluster. An administrator maintains proxy accounts using the AUTHORIZE utility. Default application accounts are maintained using the NCP utility.

6.4 Object Reuse

OpenVMS/SEVMS provides object reuse features for pages of physical memory, general purpose registers, vector processor registers (found in some VAX systems), disk files, disk and tape volumes, and printers. Other object reuse functions (removal of residual data on magnetic tapes and clearing terminal screens) are not directly supported by the OpenVMS/SEVMS TCB; the system administrator must provide procedures for enforcing object reuse on such objects.

6.4.1 Pages of Memory

OpenVMS/SEVMS memory management services (see page 87, “Memory Management”) control the relationship between physical memory and the virtual address space of a process. Following a page fault, either a page is read in from secondary storage (from a page file, swap file, image file, or global section file), or a demand-zero page is created or allocated. Demand-zero pages are initialized to zeroes when they are allocated in physical memory. OpenVMS/SEVMS global section objects are directly supported by these memory management functions. Other OpenVMS/SEVMS objects that are accessible only through system services (mailboxes, common event flag clusters, logical name tables, resource domains, and queues) are created in demand-zero pages that are within the OpenVMS/SEVMS TCB.

6.4.2 General Purpose Registers

For both Virtual Address eXtension (VAX) and Alpha processors, the hardware registers that are accessible to a process are saved and restored during a process context switch.

For VAX, the privileged instructions SVPCTX and LDPCTX are used by OpenVMS/SEVMS to save and load the process’s hardware context from the PCB. When a new process is created, the first code to execute in process context is always EXE$PROCSTRT, executing in kernel mode at Interrupt Priority Level (IPL) 2. This routine initializes the accessible registers prior to changing mode to user via a fabricated Processor Status Longword (PSL) and Program Counter (PC) and the Return from Exception or Interrupt (REI) instruction.

For Alpha, the privileged PALcode instruction SWPCTX, which is an atomic operation, returns ownership of the current Hardware Privileged Context Block (HWPCB) to the operating system and passes ownership of the new HWPCB to the processor. When a new process is created, the first code to execute in process context is always EXE$PROCSTRT, executing in kernel mode at IPL 2. This routine initializes the accessible registers prior to changing mode to user via a fabricated Processor Status (PS) and PC and the REI PALcode instruction.
6.4.3 Vector Processor Registers

The vector processor registers, which are found in some VAX (but not Alpha) systems, are not saved and restored on every process context switch. If the current process was using the vector processor, the contents of the registers remain until another process executes a vector instruction, or until the current process is scheduled again for execution. If another process executes a vector instruction, the context switch is done at that point. On a context switch, OpenVMS/SEVMS saves all of the vector registers and restores all of the vector registers of the process being scheduled.

When a process first attempts to execute a vector instruction, it will be executing on a scalar processor and will generate a vector-disabled-fault. The fault handler processing in this case includes clearing the vector processor registers. The process is then rescheduled for execution on the vector processor.

6.4.4 Disk Files

OpenVMS/SEVMS provides two features to help prevent disk scavenging: erasure patterns and highwater marking.

An erasure pattern is a repeated sequence of bits written over a file when it is deleted. The ERASE option may be specified for an entire volume, or on a per-file basis. A default of ERASE ON DELETE may be established for a volume to specify that any files created on that volume should automatically be erased when they are deleted.

Highwater marking is the system default on all volumes. Two different techniques are used to implement highwater marking. For sequential, exclusively accessed files, OpenVMS/SEVMS prevents a user process from reading disk blocks beyond those that have been written. For indexed files and shared sequential files, an erase-on-allocate technique is used: disk blocks are erased when they are allocated to a file. Highwater marking for a volume can be disabled by a privileged user.

OpenVMS/SEVMS disk files are defined by the set of disk blocks that are allocated to the file. If a user rewrites part of a file, but does not write over all of the previously written data blocks, then the data in those blocks will still exist in the file and be accessible to any subject authorized to read the file. However, no subject can read beyond the last block of user-written data.

6.4.5 Disk and Tape Volumes

When an OpenVMS/SEVMS disk or tape volume is initialized, the ERASE parameter can be used to erase the entire volume before writing the volume labels. When SEVMS erases a tape volume, it first leaves the existing volume label, then erases the rest of the volume, and finally overwrites the original label with the new label. The TFM requires that, for the evaluated configuration, tape volumes must be erased prior to reassignment to a different user.

6.4.6 Printers

The three ANSI printer protocols supported by OpenVMS/SEVMS all provide a reset control sequence that can be used to reset the printer to a known initial state. When a print queue is initialized, the SEPARATE
6.5 Assurances

This section provides an overview of the assurances developed by Digital to support the OpenVMS/SEVMS TCB. The Digital testing procedures for both hardware and software are discussed, followed by a description of the BI informal security policy model. The Digital RAMP plan and security analysis procedures are also discussed.

6.5.1 Hardware Testing

All processors (VAX and Alpha) and peripherals in the evaluated configuration are identified within Appendix A of this report. During the entire life cycle of a system, Digital performs extensive hardware testing to assure that the system operates correctly. This section provides an overview of the testing performed during the manufacturing process, the testing available on-site at customer locations, and the testing supported by OpenVMS/SEVMS.

6.5.1.1 VAX Conformance Tests

For VAX systems, Digital's architecture verification testing is based on two primary programs: Architecture Exerciser (AXE) and Multi-instruction Architecture Exerciser (MAX). These tests are used during processor design as well as during manufacturing. The AXE and MAX tests were developed over a period of several years by Digital's Architecture Verification Group. This group maintains and enhances the architecture conformance tests.

AXE and MAX divide an instruction's context into several components; and, for each component, valid and invalid values are pseudo-randomly selected to create a test case. The exerciser creates unique test cases for as long as it is allowed to run. The minimum number of test cases of AXE and MAX are chosen based on the complexity of the particular processor design. Prior to volume manufacturing, there must be successful completion of a predetermined number of AXE and MAX test cases. During the manufacturing of individual system components and during the integration of the components into a complete VAX system, additional hardware testing is performed at each phase of assembly.

6.5.1.2 Alpha Conformance Tests

Digital's architecture verification testing for Alpha processors is based on the RISC/Alpha AXP Architecture Exerciser (RAX) program. RAX tests are used during processor design as well as during manufacturing. The RAX tests were developed by Digital's Architecture Verification Group. This group maintains and enhances the architecture conformance tests.

RAX tests divide an instruction's context into several components; and, for each component, valid and invalid values are pseudo-randomly selected to create a test case. The exerciser creates unique test cases.
for as long as it is allowed to run. Prior to volume manufacturing, there must be successful completion of a predetermined number of RAX test cases. During the manufacturing of individual system components and during the integration of the components into a complete Alpha system, additional hardware testing is performed at each phase of assembly.

6.5.1.3 On-Site Tests (VAX)

When electrical power is applied to a VAX, a series of power-up tests is run automatically. These tests check each component in the system and inform the operator about the success or failure of the tests. In addition, each VAX console subsystem provides an interface to execute predefined diagnostics and/or to load additional diagnostics from tape or disk. More extensive hardware testing, which involves VAX instruction set testing, can be performed at customer sites by Digital. For larger VAX systems, these tests would routinely be run by Digital’s Field Service organization as part of installation and maintenance. These more extensive diagnostics include testing of memory management, hardware access modes, and privileged instructions.

6.5.1.4 On-Site Tests (Alpha)

When electrical power is applied to an Alpha processor, a series of power-up tests is run automatically. These tests check each component in the system and inform the operator about the success or failure of the tests. In addition, each Alpha console subsystem provides an interface to execute additional predefined tests. More extensive hardware testing, which involves Alpha instruction set testing, can be performed at customer sites by Digital.

6.5.1.5 VMS Supported Tests

Software tools associated with OpenVMS/SEVMS are also used to report hardware errors and to perform some hardware testing. These tools include the User Environment Test Package (UETP), the error logging facility (ERRFMT), and the Operator Communication Manager (OPCOM) process.

The UETP is an exerciser that simulates users’ running programs. The exerciser checks for file system inconsistencies, and performs some diagnostics on selected devices. UETP does not diagnose the cause of a hardware error; it simply reports the error and continues with the next test. Hardware error messages are automatically recorded in the OpenVMS/SEVMS error log file by the ERRFMT system process. The contents of the error log file can be examined after it is processed by the Error Log Report Formatter (ERF) utility. The OPCOM process displays and records messages from I/O device drivers describing the status of particular hardware devices.

6.5.2 Software Testing

Digital has an extensive, multistage testing strategy that is comprised of security, functional, and integration testing. The testing is divided into four groups: system testing, security testing, hardware testing, and layered product testing. A comprehensive test suite has been created for security testing. The Digital test plan[3] describes the test approach, the hardware and software requirements, and the functions that are tested. The testing approach uses various methods to provide TCB coverage. These methods involve black
and white box testing, as well as monolithic system testing. The security test suite focuses on the security mechanisms that execute in kernel mode, but includes DCL commands (those that perform functions beyond merely translating commands and command line parameters into system service calls). The test suite covers MAC, DAC, identification and authentication, object reuse, auditing, privileges, security administration, and privileged interfaces. The entire security functional test suite is run against the final integrated releases of OpenVMS and SEVMS.

System testing is performed in the following manner. A selected subset of all the OpenVMS/SEVMS tests are run to test the new functions developed for the particular release and to ensure that no previously existing functions have been altered. The tests are run in stages, ranging from small areas of the system to tests that exercise many of the system’s functions. A base level, or internal version, of the system is established.

During testing, a revision test matrix is created that tracks changes to the system among the development groups. Test procedures are then developed based on the matrix to cover the major components or subsystems that are affected by the latest set of changes. Some of the tests can be run automatically; others are manual (e.g., fault tests). In addition, the system is “field” tested, both at internal and external sites. Field testing involves exercising the system on as many different types of configurations as possible. The tests performed during field testing are chosen by site personnel. Finally, application testing of Digital’s layered products is performed at both Digital and customer facilities.

### 6.5.3 Informal Security Policy Model

The **VAX SEVMS Security Model**[6] document describes the security policy enforced by SEVMS. Since the discretionary policy of OpenVMS is identical to the discretionary policy of SEVMS, the informal model also provides a detailed description of the policy enforced by OpenVMS.

The informal model first describes the security attributes of the subjects and objects defined in SEVMS. The types of access that subjects can make to objects are described in general terms.

The model defines four security properties enforced by the SEVMS TCB: the simple security property, the star property, the discretionary security property, and the tranquility property. The remainder of the model discusses how SEVMS enforces these four properties for the subjects and objects it defines.

For each type of object, the model presents the following topics:

1. Description of the object
2. How the object is named
3. The use of ranges of classification
4. How the object is implemented in a cluster environment
5. The object’s persistence across system startups
6. Discretionary access types supported
7. Mandatory access controls for the object
8. Descriptions of the system services that mediate access to the object
9. Descriptions of object creation and initial access attributes
10. Object deletion
11. Special considerations.

The final section of the model relates the SEVMS entities to the constructs used in the Bell-LaPadula security model. This section correlates the discretionary and mandatory access types of SEVMS to one another and to the Bell-LaPadula access attributes. It also explains, for each object type, how “access” is defined in the context of OpenVMS/SEVMS data structures.

6.5.4 Rating Maintenance Phase


Digital’s RM-Plan consists of seven chapters and ten appendixes. Chapter and appendix designators and their titles are as follows:

Chapter 1 Introduction
Chapter 2 Responsible Individuals
Chapter 3 OpenVMS and SEVMS Configuration Items
Chapter 4 Engineering Process and Security Analysis
Chapter 5 RAMP Assurance and Evidence
Chapter 6 RAMP Audits
Chapter 7 Security Incidents
Appendix A Facility List
Appendix B Example Facility
Appendix C V6.0 Project Questionnaire
Appendix D V6.0 Readiness Review Questionnaire
Appendix E V6.0 Source Change Tracking (SCT) Template
Appendix F RAMP Requirements (from EvalAnnouncements)
Appendix G Mapping of RAMP Requirements to RM-Plan
Appendix H DEC Glossary
Appendix I Hardware List
Appendix J Roadmap References

Chapter 1 of the RM-Plan provides an introduction to NSA’s RAMP program, identifies applicable government documents, and contains pointers to Appendixes F and G. The purpose of Chapter 2 is to identify those individuals at Digital who fill the RAMP roles of Vendor Business Point of Contact (VBPOC), Responsible Corporate Officer (RCO), and Vendor Security Analyst (VSA).

Chapter 3 of the RM-Plan identifies the configuration items for OpenVMS and SEVMS. Digital’s approach to TCB software configuration items employs the concept of a facility. A facility is a single configuration item that consists of a number of related source files. Each facility affects one or more executable modules of OpenVMS/SEVMS. Appendix A of the RM-Plan lists each facility, and Appendix B provides specifics about an example facility, the MOUNT facility. Appendix I contains the list of configuration items for the
hardware components of the TCB. For hardware, whenever new components and models are introduced, the hardware list is updated and VSAs perform security analysis of the new hardware. Documentation configuration items include user documentation, design documentation, and test documentation. Some of the documentation is maintained by the VSAs while other documents are managed by different Digital organizations. Appendix J contains the list of references cited by the primary design document, the Design Documentation Roadmap[8].

Chapter 4 of the RM-Plan, “Engineering Process and Security Analysis,” provides a discussion of Digital’s engineering process and details how security analysis is performed. The RAMP procedures for SEVMS are somewhat different than those for OpenVMS since SEVMS is maintained by a smaller engineering staff. However, the basic approach for security analysis and the generation of RAMP evidence is the same. The Digital VSAs work with the OpenVMS/SEVMS engineering staff to identify security-relevant changes and to assure that appropriate security analysis is performed. The forms described in Appendixes C, D, and E of the RM-Plan facilitate the RAMP procedures during the engineering process.

For both OpenVMS and SEVMS, a change is either a bug fix or a major development effort, referred to by Digital as a project. For each project included in a release, security analysis is performed and documented in conjunction with Digital’s release integration process. The integration process includes design reviews, project readiness reviews, and source change tracking. Digital’s VSAs participate in the review processes and document their analysis as an integral part of the release documentation. The VSAs also use project design documentation as the basis for updating the security test suite and for monitoring revisions to the RAMP documentation configuration items.

If a change is the result of a bug fix, security analysis is performed based on source change tracking information rather than project design documentation. All bug fixes are reviewed for security relevance. Any bug fix identified as security-relevant is analyzed by a VSA prior to approval for inclusion in the release. In this case, the security analysis is documented with the source change tracking information related to the bug fix.

Chapter 5 of Digital’s RM-Plan identifies the RAMP evidence that is generated and maintained. This evidence includes the following:

- Results of OpenVMS project design and readiness reviews (includes security analysis for the project)
- The OpenVMS SCT database for the release (includes bug fix security analysis)
- The SEVMS SCT database for the release (includes security analysis documentation)
- Updated test documentation (when necessary)
- A backup of the Security Test Suite used to generate test evidence for the release
- OpenVMS source code for the release
- SEVMS source code for the release
- New and revised design documentation, including an updated design documentation roadmap

Chapter 6 of the RM-Plan discusses how VSA-conducted audits of OpenVMS/SEVMS are performed. Chapter 7 provides a discussion of how Digital responds to security incidents, and identifies Digital’s Software Security Response Team (SSRT) as responsible for the management of all security incidents.
This page intentionally left blank
Part III

TCSEC Evaluation Requirements
Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha
Chapter 7
Evaluation as a C2 System

7.1 Discretionary Access Control

Requirement

The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals, or defined groups of individuals, or by both, and shall provide controls to limit propagation of access rights. The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorized access. These access controls shall be capable of including or excluding access to the granularity of a single user. Access permission to an object by users not already possessing access permission shall only be assigned by authorized users.

Applicable Features

The OpenVMS Trusted Computing Base (TCB) provides two complementary discretionary access control mechanisms: Access Control List (ACL) entries and User Identification Code (UIC) protection fields. All OpenVMS named objects have protection fields that specify access types for each of four categories of users: SYSTEM, GROUP, OWNER, and WORLD. In addition to a username, each OpenVMS subject has a UIC that is compared to the owner UIC of the object to determine the subject’s category.

An ACL can be specified for all OpenVMS named objects except tape volumes and files. Each entry in the ACL specifies authorized access types for one or more identifiers. Each subject in OpenVMS has a set of authorized identifiers associated with the user’s UIC. If an ACL entry for an object does not explicitly grant or deny a subject the requested access, then the UIC protection fields are used to determine access.

The OpenVMS Trusted Facility Manual (TFM) states that each username must be associated with a unique UIC to conform to the C2 requirements to provide access controls to the granularity of a single user.

All named objects require CONTROL access to change the discretionary protection data (ACL and/or protection fields) for the object; by default, the creator of an object has CONTROL access. For each type of named object except files, site-specific default protection fields and ACL entries can be defined for use when new objects are created; Digital provides an initial profile for each type of object. For disk files, a number of different mechanisms exist to specify initial protection fields and ACL entries.

When discretionary protection data for an object is changed, current accesses are not affected; all subsequent access mediation decisions are based on the new protection data.

The specific named objects defined by OpenVMS are discussed in detail starting on page 143, “Objects”. For
more information on the discretionary access control mechanisms of OpenVMS see page 155, “Discretionary Access Control”.

Conclusion

OpenVMS satisfies the C2 Discretionary Access Control requirement.

Additional Requirement (B3)

CHANGE: The enforcement mechanism (e.g., access control lists) shall allow users to specify and control sharing of those objects, and shall provide controls to limit propagation of access rights. These access controls shall be capable of specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective modes of access to that object.

ADD: Furthermore, for each such named object, it shall be possible to specify a list of named individuals and a list of groups of named individuals for which no access to the object is to be given.

Applicable Features

The OpenVMS ACL mechanism supports the granting or denying of access to the granularity of a single user. In addition, the identifier mechanism supports the definition of groups of users to which access can be granted or denied.

Conclusion

OpenVMS satisfies the B3 Discretionary Access Control requirement.1

7.2 Object Reuse

Requirement

All authorizations to the information contained within a storage object shall be revoked prior to initial assignment, allocation or reallocation to a subject from the TCB’s pool of unused storage objects. No information, including encrypted representations of information, produced by a prior subject’s actions is to be available to any subject that obtains access to an object that has been released back to the system.

1 Although OpenVMS satisfies this B3 functional requirement, it does not satisfy the assurance requirements above its rated level.
Applicable Features

The OpenVMS storage objects that can contain residual data are: memory pages, disk blocks, hardware registers, and magnetic tapes. When a page of physical memory is allocated to a subject, it is either cleared to all zeroes (demand-zero page) or read in from secondary storage (for instance, from a page file).

Disk blocks are allocated to a subject when a file is created or extended. For non-shared files being accessed sequentially, OpenVMS maintains a highwater mark of the highest logical block written to the file; read access to logical blocks greater than the highwater mark is not allowed by the TCB. For other files, an erase-on-allocate option is the default; this option writes an erasure pattern when a block is allocated to the file. Refer to page 175, “Object Reuse”, for further details.

The general purpose hardware registers accessible to untrusted subjects are loaded or cleared by OpenVMS during a process context switch. Vector processor hardware registers, found in some Virtual Address eXtension (VAX) processors, are only loaded or cleared when a new process requires the use of the vector processor. If there has been no intervening use of the vector processor registers, a complete context switch is not performed.

OpenVMS provides an option to erase a disk or tape volume during volume initialization. This feature can be used by an administrator to write an erasure pattern on a volume. For a C2 environment, the OpenVMS TFM requires that a volume be erased prior to reassignment to a different user.

For more details on the object reuse features of OpenVMS, refer to page 175, “Object Reuse”.

Conclusion

OpenVMS satisfies the C2 Object Reuse requirement.

7.3 Identification and Authentication

Requirement

The TCB shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall use a protected mechanism (e.g., passwords) to authenticate the user’s identity. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user. The TCB shall be able to enforce individual accountability by providing the capability to uniquely identify each individual ADP system user. The TCB shall also provide the capability of associating this identity with all auditable actions taken by that individual.

Applicable Features

OpenVMS provides support for passwords for all users; the applicable TFM describes how to assure that all login accounts require passwords to conform to the C2 evaluated configuration. The authentication data for a user is stored within the OpenVMS security database (SYSUAF.DAT) and is protected from unauthorized
access by OpenVMS discretionary access control mechanisms. User passwords are encrypted within the database. However, when passwords are used with DECnet applications, the passwords are sent over the DECnet communication media unencrypted.\(^2\)

Each user is uniquely identified by username; the audit records generated by the TCB contain the username in all user activity records. Since all nodes in a VMScluster use the same security database, the identity of a user is the same on all nodes.

Conclusion

OpenVMS satisfies the C2 Identification and Authentication requirement.

7.4 Audit

Requirement

The TCB shall be able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data. The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into a user’s address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user’s address space and for object deletion events the audit record shall include the name of the object. The ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity.

Applicable Features

OpenVMS provides the capability to audit certain specified security-related events and store them in an audit log and, optionally, an archive file. The log and archive files are protected such that only those possessing the appropriate access can manipulate their contents. The system is capable of auditing identification and authorization events such as logins and DECnet connection establishment. The system can also audit object-related functions such as open, read, write, and delete. Operator and system administrator activities can also be audited.

All audit records contain the date and time of an event, the type of event, and the associated username and process ID. For identification and authorization events, the source of the request (terminal ID or remote node) is included in the generated record. For object accesses, the audit record includes the type of object and its name.

\(^2\)In an OpenVMS evaluated configuration, all of the TCB hardware must be physically protected.
The ANALYZE/AUDIT utility gives the user or administrator a powerful tool with which to select records from the audit log. Searches can be based on use of a particular privilege, on a user, or on an object security level.

Conclusion

OpenVMS satisfies the C2 Audit requirement.

7.5 System Architecture

Requirement

The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures). Resources controlled by the TCB may be a defined subset of the subjects and objects in the ADP system. The TCB shall isolate the resources to be protected so that they are subject to the access control and auditing requirements.

Applicable Features

OpenVMS includes features that protect it from external interference, provide address space isolation, and assure that access mediation and audit are performed for all protected resources.

OpenVMS employs primarily memory management services to protect the TCB from interference. A processor supports four hierarchical hardware access modes for process execution. The modes are, from least privileged to most privileged: user, supervisor, executive, and kernel. For VAX processors, the current mode of execution is kept in the Processor Status Longword (PSL), and for Alpha the current mode of execution is kept in the Processor Status (PS) register. Memory protection is based on hardware access modes; page protection codes are contained in the Page Table Entry (PTE) for each virtual page. For each hardware access mode, three types of access are defined: no access, read-only, and read-write. A page of memory accessible in a particular mode is also accessible in all of the more privileged modes. During address translation, the processor raises an access-control-violation fault if the current mode of execution is not authorized to access the page. TCB software establishes and maintains the page protection codes. Within the PTE an “owner” access mode is maintained by TCB software; changes to the protection code can only be made in an access mode at least as privileged as the owner access mode.

Access modes are used by TCB software to protect logical TCB data structures as well as pages of memory. For example, any logical names the TCB uses to resolve references to TCB data files will be defined by the TCB in a protected mode so they cannot be redefined by untrusted subjects. Record Management Services (RMS) uses access mode protection for resource locks to separate its resources from the resources managed by untrusted subjects. In general, logical data structures that can be created both by untrusted subjects and by the TCB will have associated access mode protection to isolate the TCB data structures.

OpenVMS TCB data that is stored in files on disk is protected by the OpenVMS discretionary access control mechanisms and is generally accessible only to subjects in the SYSTEM category. Other TCB
Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha

CHAPTER 7. EVALUATION AS A C2 SYSTEM

data is protected by the use of OpenVMS privileges; for example, a subject with the SYSNAM privilege is authorized to change the system directory of logical name tables.

The protected resources in OpenVMS are devices, volumes, disk and tape files, and a number of OpenVMS-specific objects that can be used for interprocess communication (global sections, resource domains, common event flag clusters, logical name tables, and queues). The OpenVMS TCB controls all access to protected objects using system services and privileged images.

System services are implemented using the change mode instructions to transfer control from an untrusted program to the TCB. The change mode instructions transfer control to OpenVMS change mode dispatchers (via the System Control Block (SCB)) and provide a numeric parameter. The execution access mode is raised to the specified mode; and the dispatcher, based on the parameter value, transfers control in the resulting access mode. The mode may only be increased; otherwise, the change mode instruction does not change the execution mode of the process. The Return from Exception or Interrupt (REI) instruction is used by the TCB to return control to the user program. OpenVMS privileged images can be invoked by an untrusted subject to perform security-relevant operations. Each OpenVMS privileged image performs specific operations on behalf of the user; when the image terminates, the privileges associated with the image are removed from the process. All security-relevant operations on protected resources are performed on behalf of an untrusted subject by OpenVMS system services and privileged images.

OpenVMS has a distributed architecture. Each node in a VMScluster configuration executes the TCB code independently. Local and cluster-visible resources are protected by the local TCB code. Access via DECnet software to non-cluster-visible resources are protected by the TCB code on the node where the resource is located. The security information upon which the policy is based is kept in a single security database, thereby enforcing a uniform security policy.

All operations on objects protected by the TCB are auditable by the TCB code that protects the resource. In addition to object creation, access, and deletion, events such as logins, logouts, login failures, changes to the security database, use of privileges, volume mounts and dismounts, and network management changes are auditable by the TCB. For a complete list of the types of auditable events, see page 163, “Auditing”.

Conclusion

OpenVMS satisfies the C2 System Architecture requirement.

Additional Requirement (B1)

ADD: The TCB shall maintain process isolation through the provision of distinct address spaces under its control.

Applicable Features

OpenVMS provides process address space isolation through its virtual memory management. Each process has a unique, virtual address space enforced through OpenVMS paging mechanisms. The process virtual address space is divided into several logical regions. The system region, in which the system executive
resides, is global to all processes; page protection is used to prevent unauthorized access. There are two process-unique regions that contain control structures and code and data, respectively.

Conclusion

OpenVMS satisfies the B1 System Architecture requirement.

7.6 System Integrity

Requirement

Hardware and/or software features shall be provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the TCB.

Applicable Features

VAX Systems

All VAX systems are supplied with a set of hardware diagnostics that are executed during power-on processing. These diagnostics check the basic hardware for proper operation and include tests of processor microcode, data path circuitry, and, in some processors, self-tests of attached devices. VAX hardware performs a number of consistency checks throughout normal operation, including processor checks and error correction functions.

In addition to the power-on diagnostics, Digital supports several diagnostic packages that are available through Digital Field Service. These packages include tests of virtual memory management, hardware access modes, page protection, privileged instructions, and the change mode instructions. They also include diagnostics for the attached hardware that can be configured with each processor.

Alpha Systems

All Alpha systems are supplied with a set of hardware tests that are executed during power-on processing. These tests check the basic hardware for proper operation and validate data path circuitry as well as system interconnects to memory and I/O devices. In addition to the power-on tests, Digital supports a test package, available through Digital Field Service, that tests the Alpha CPU. Alpha processors also provide extended console-selectable tests and exercizers for memory and I/O devices, and Alpha hardware performs a number of consistency checks throughout normal operation, including processor checks and error correction functions.

OpenVMS provides the User Environment Test Package (UETP) which is a system exerciser designed to stress the system by causing the system to perform all required functions. UETP can be configured to test any or all areas of the system. These include architectural features; peripherals; load testing the system; network connections; memory; and cluster connections.

Many peripherals also have on-board tests that can be used to verify correct operation, thus allowing for additional verification of any attached device.
Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha
CHAPTER 7. EVALUATION AS A C2 SYSTEM

Conclusion

OpenVMS satisfies the C2 System Integrity requirement.

7.7 Security Testing

Requirement

The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. Testing shall be done to assure that there are no obvious ways for an unauthorized user to bypass or otherwise defeat the security protection mechanisms of the TCB. Testing shall also include a search for obvious flaws that would allow violation of resource isolation, or that would permit unauthorized access to the audit or authentication data.

Applicable Features

Digital’s security test suite has evolved since OpenVMS was initially evaluated. For the current RAMP Action, additional tests were added to the security test suite that was used for the RAMP of OpenVMS VAX Version 6.1, and the security test suite was modified to run on both VAX and Alpha systems.

The security test suite was successfully executed at a Digital facility by Digital VSAs on a Mixed Architecture VMScluster (MAVC) system within a common security domain. The MAVC consisted of a VAX 8800, VAX 4000, AlphaServer 2100 and a DEC 7000.

Conclusion

OpenVMS satisfies the C2 Security Testing requirement.

7.8 Design Specification and Verification

Requirement

None.

Additional Requirement (B1)

NEW: An informal or formal model of the security policy supported by the TCB shall be maintained over the life cycle of the ADP system and demonstrated to be consistent with its axioms.
Applicable Features

Digital has provided an informal security policy model that describes the OpenVMS subjects and objects, and the discretionary security properties enforced by the OpenVMS TCB. For each protected object, the model shows how the TCB assures that the security properties are maintained. The model is described on page 179, “Informal Security Policy Model”.

Conclusion

OpenVMS satisfies the B1 Design Specification and Verification requirement.

7.9 Security Features User’s Guide

Requirement

A single summary, chapter, or manual in user documentation shall describe the protection mechanisms provided by the TCB, guidelines on their use, and how they interact with one another.

Applicable Features

Chapters 1 through 5 of the Open VMS Guide to System Security[34] constitute the OpenVMS Security Features User’s Guide (SFUG). These chapters apply to both VAX and Alpha systems. The first portion (Chapters 1 and 2) of the guide describes the general nature of security problems and presents an overview of the security policy. Chapter 3 covers user security on a single node and in a cluster environment. The intent of this chapter is to recommend good practices regarding the use of accounts and passwords. The topic of auditing is also discussed. Chapters 4 and 5 discuss the security objects and the associated protection mechanisms. Guidelines concerning management of the access control lists and default file access are also presented. The protection mechanisms are used to ensure that a requesting subject has the credentials (rights identifiers, hardware access modes, privileges) to access an object in the manner specified in the object’s ACL.

Conclusion

CHAPTER 7. EVALUATION AS A C2 SYSTEM

7.10 Trusted Facility Manual

Requirement

A manual addressed to the ADP system administrator shall present cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given.

Applicable Features

VAX Systems

The OpenVMS TFM is composed of Chapters 6 to 13 and Appendices A to D of the OpenVMS Guide to System Security[34], the “Audit Analysis” section of the OpenVMS System Management Utilities Reference Manual[25][26], the OpenVMS VAX Version 6.1 Upgrade and Installation Manual[38], and the documentation changes to these manuals described in OpenVMS VAX Version 6.1 Release Notes[36]. Chapters 6-8 of the guide to system security describes how to register users and initialize and change the user’s security characteristics; Chapter 9 describes the auditing facilities of OpenVMS; Chapter 10 discusses actions to be taken when specific security problems arise; Chapter 11 discusses issues specific to a VAXcluster environment; Chapter 12 discusses DECnet security concerns, and Chapter 13 describes protected subsystems and explains how to build them. Appendix A lists all of the OpenVMS privileges, and provides details of the specific operations that each privilege authorizes. Appendix C provides information on how to configure OpenVMS to conform to the C2 requirements of the evaluated configuration.

The upgrade and installation manual describes how to install a new version of OpenVMS, including specific instructions to assure that the installed version conforms to the requirements for the evaluated configuration. The “Audit Analysis” section of the utilities manual provides a detailed description of the audit events and the data that is recorded for each event, as well as a description of how to analyze the OpenVMS audit data. Additional sections of the utilities manual provide details of the commands and operations discussed in the guide to system security.

Alpha Systems

The OpenVMS TFM is composed of Chapters 6 to 13 and Appendices A to D of the OpenVMS Guide to System Security[34], the “Audit Analysis” section of the OpenVMS System Management Utilities Reference Manual, V6.1[25][26], the OpenVMS Version 6.1 Installation Guide[37], and the documentation changes to these manuals described in OpenVMS AXP Version 6.1 Release Notes[14]. Chapters 6-8 of the guide to system security describes how to register users and initialize and change the user’s security characteristics; Chapter 9 describes the auditing facilities of OpenVMS; Chapter 10 discusses actions to be taken when specific security problems arise; Chapter 11 discusses issues specific to a VM-Schuster environment; Chapter 12 discusses DECnet security concerns, and Chapter 13 describes protected subsystems and explains how to build them. Appendix A lists all of the OpenVMS privileges, and provides details of the specific operations that each privilege authorizes. Appendix C provides information on how to configure OpenVMS to conform to the C2 requirements of the evaluated configuration.

The upgrade and installation manual describes how to install a new version of OpenVMS, including specific instructions to assure that the installed version conforms to the requirements for the evaluated configuration. The “Audit Analysis” section of the utilities manual provides a detailed description of the audit events and
the data that is recorded for each event, as well as a description of how to analyze the OpenVMS audit data. Additional sections of the utilities manual provide details of the commands and operations discussed in the guide to system security.

Conclusion

OpenVMS satisfies the C2 Trusted Facility Manual requirement.

7.11 Test Documentation

Requirement

The system developer shall provide to the evaluators a document that describes the test plan, test procedures that show how the security mechanisms were tested, and results of the security mechanisms’ functional testing.

Applicable Features

The Digital test plan, B1 Test Strategy for VMS and SEVMS[3], addresses the hardware and software requirements for executing the test suite, the overall testing approach, and the specific tests that are in the test suite. The basic approaches used are black box testing (functional testing of code without knowledge of what the code is doing), white box testing (functional testing of code with knowledge of how the code executes), and monolithic testing (testing all combinations of variables against a system). The monolithic and black box testing approaches are used to test Discretionary Access Control (DAC), and white box testing is used to test system services, Digital Command Language (DCL) commands, utilities, object reuse, login, auditing, privilege use, and cluster operation.

The test suite is designed to execute a representative set of the possible access mediation scenarios to sufficiently demonstrate that the security decision functions are implemented as intended. That is,

Either the system service or the associated DCL command that calls the service is used as the test interface. In some cases, both the command and the system service are tested. All operations involving access mediation are tested, including those operations requiring use of privilege. Combinations of subject and object access are used to test the access mediation mechanisms. Each of the possible access types for each object class are tested with subjects having different security characteristics. Testing of privileged operations is performed from the perspective of preserving the TCB boundary. That is, testing provides a measure of assurance that operations requiring privilege cannot be performed by subjects that do not possess the privilege. Other security mechanisms, such as the audit subsystem and identification and authentication features, are also covered by the test suite.

Digital provided evidence of the successful execution of the entire C2 security test suite in a VMSccluster configuration.
CHAPTER 7. EVALUATION AS A C2 SYSTEM

Conclusion

OpenVMS satisfies the C2 Test Documentation requirement.

7.12 Design Documentation

Requirement

Documentation shall be available that provides a description of the manufacturer’s philosophy of protection and an explanation of how this philosophy is translated into the TCB. If the TCB is composed of distinct modules, the interfaces between these modules shall be described.

Applicable Features

VAX Systems

Digital has provided a Design Documentation Roadmap[8] that serves as an index to the security-relevant information in their extensive design documentation. This roadmap is built with an outline similar to the outline of this document. The roadmap points to sections of a number of secondary references. The list of secondary references to the design document are listed in the references to this report. These references describe the internal design of the system. Some modules of the system are documented very precisely while others are more high level. All of the TCB including all of the hardware components are documented to some degree.

The references of [8] are listed in the “References” appendix of this report. Several of the more important references are the VMS Internals and Data Structures: Version 5.2[80], VAXcluster Principles[74], VMS File System Internals[79], and VAX Architecture Reference Manual[93]. [80], [74], and [79] cover the basic software architecture of OpenVMS and SEVMS, Version 6.1, with VAX or Alpha. [93] contains the basic hardware architecture. These references together form the basis of description of the TCB design.

Alpha Systems

Digital has provided an updated OpenVMS/SEVMS VAX and Alpha V6.1 Design Documentation Roadmap[8] that serves as an index to the security-relevant information in their extensive design documentation. This roadmap is built with an outline similar to the outline of this document. The roadmap points to sections of a number of secondary references. The list of secondary references to the design document are listed in the references to this report. These references describe the internal design of the system. Some modules of the system are documented very precisely while others are more high level. All of the TCB including all of the hardware components are documented to some degree.

The references of [8] are listed in the “References” appendix of this report. Several of the more important references are OpenVMS AXP Internals and Data Structures, Version 1.5[72], Guidelines for VMScluster Configurations[13], VMScluster Systems for OpenVMS[40], VMS File System Internals[79], and Alpha Architecture Reference Manual[83]. [72], [13], [75], and [79] cover the basic software architecture of OpenVMS and SEVMS, Version 6.1, with VAX or Alpha. [83] contains the basic
hardware architecture. These references together form the basis of description of the TCB design.

Conclusion

OpenVMS satisfies the C2 Design Documentation requirement.

Additional Requirement (B1)

_ADD_: An informal or formal description of the security policy model enforced by the TCB shall be available and an explanation provided to show that it is sufficient to enforce the security policy. The specific TCB protection mechanisms shall be identified, and an explanation given to show that they satisfy the model.

Applicable Features

The informal security policy model [6] provides an explanation of how OpenVMS enforces the security policy. For each protected object, the model identifies the TCB mechanisms (e.g., system services) that enforce the policy for that object.

Conclusion

OpenVMS satisfies the B1 Design Documentation requirement.
CHAPTER 7. EVALUATION AS A C2 SYSTEM

This page intentionally left blank
Chapter 8
Evaluation as a B1 System

8.1 Discretionary Access Control

Requirement

The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals, or defined groups of individuals, or by both, and shall provide controls to limit propagation of access rights. The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorized access. These access controls shall be capable of including or excluding access to the granularity of a single user. Access permission to an object by users not already possessing access permission shall only be assigned by authorized users.

Applicable Features

The SEVMS Trusted Computing Base (TCB) provides two complementary discretionary access control mechanisms: Access Control List (ACL) entries and User Identification Code (UIC) protection fields. All SEVMS named objects have protection fields that specify access types for each of four categories of users: SYSTEM, GROUP, OWNER, and WORLD. In addition to a username, each SEVMS subject has a UIC that is compared to the owner UIC of the object to determine the subject’s category.

An ACL can be specified for all SEVMS named objects except tape volumes and files. Each entry in the ACL specifies authorized access types for one or more identifiers. Each subject in SEVMS has a set of authorized identifiers associated with the user’s UIC. If an ACL entry for an object does not explicitly grant or deny a subject the requested access, then the UIC protection fields are used to determine access.

The SEVMS Trusted Facility Manual (TFM) states that each username must be associated with a unique UIC to conform to the B1 requirements to provide access controls to the granularity of a single user.

All named objects require CONTROL access to change the discretionary protection data (ACL and/or protection fields) for the object; by default the creator of an object has CONTROL access. For each type of named object except files, site-specific default protection fields and ACL entries can be defined for use when new objects are created; Digital provides an initial profile for each type of object. For disk files, a number of different mechanisms exist to specify initial protection fields and ACL entries.

When discretionary protection data for an object is changed, current accesses are not affected; all subsequent access mediation decisions are based on the new protection data.

The specific named objects defined by SEVMS are discussed in detail starting on page 143, “Objects”. For
more information on the discretionary access control mechanisms of SEVMS see page 155, “Discretionary Access Control”.

Conclusion

Security Enhanced VMS (SEVMS) satisfies the B1 Discretionary Access Control requirement.

Additional Requirement (B3)

CHANGE: The enforcement mechanism (e.g., access control lists) shall allow users to specify and control sharing of those objects, and shall provide controls to limit propagation of access rights. These access controls shall be capable of specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective modes of access to that object.

ADD: Furthermore, for each such named object, it shall be possible to specify a list of named individuals and a list of groups of named individuals for which no access to the object is to be given.

Applicable Features

The SEVMS ACL mechanism supports the granting or denying of access to the granularity of a single user. In addition, the identifier mechanism supports the definition of groups of users to which access can be granted or denied.

Conclusion

SEVMS satisfies the B3 Discretionary Access Control requirement.¹

8.2 Object Reuse

Requirement

All authorizations to the information contained within a storage object shall be revoked prior to initial assignment, allocation or reallocation to a subject from the TCB’s pool of unused storage objects. No information, including encrypted representations of information, produced by a prior subject’s actions is to be available to any subject that obtains access to an object that has been released back to the system.

¹Although SEVMS satisfies this B3 functional requirement, it does not satisfy the assurance requirements above its rated level.
Applicable Features

The SEVMS storage objects that can contain residual data are: memory pages, disk blocks, hardware registers, and magnetic tapes. When a page of physical memory is allocated to a subject, it is either cleared to all zeroes (demand-zero page) or read in from secondary storage (for instance, from a page file).

Disk blocks are allocated to a subject when a file is created or extended. For non-shared files being accessed sequentially, SEVMS maintains a highwater mark of the highest logical block written to the file; read access to logical blocks greater than the highwater mark is not allowed by the TCB. For other files, an erase-on-allocate option is the default; this option writes an erasure pattern when a block is allocated to the file. Refer to page 175, “Object Reuse”, for further details.

The general purpose hardware registers accessible to untrusted subjects are loaded or cleared by SEVMS during a process context switch. Vector processor hardware registers, found in some Virtual Address eXtension (VAX) processors, are only loaded or cleared when a new process requires the use of the vector processor. If there has been no intervening use of the vector processor registers, a complete context switch is not performed.

SEVMS provides an option to erase a disk or tape volume during volume initialization. This feature can be used by an administrator to write an erasure pattern on a volume. For a Bl environment, the SEVMS TFM requires that a volume be erased prior to reassignment to a different user.

For more details on the object reuse features of SEVMS, refer to page 175, “Object Reuse”.

Conclusion

SEVMS satisfies the Bl Object Reuse requirement.

8.3 Labels

Requirement

Sensitivity labels associated with each subject and storage object under its control (e.g., process, file, segment, device) shall be maintained by the TCB. These labels shall be used as the basis for mandatory access control decisions. In order to import non-labeled data, the TCB shall request and receive from an authorized user the security level of the data, and all such actions shall be auditable by the TCB.

Applicable Features

Every identified storage object in SEVMS has an associated classification label that is maintained by the TCB. Each process has an associated clearance label that is constrained to be within the user’s maximum and minimum clearance authorizations. These labels are used by the TCB to enforce the SEVMS mandatory access control policy.

Unlabeled data may be imported from unlabeled magnetic tape volumes. A subject importing data from an
unlabeled tape volume must have the IMPORT privilege and must specify an explicit classification label for
the data. The use of the IMPORT privilege is auditable. Other unlabeled data must be imported via single
level devices; the TCB uses the label assigned to the device as the label for the data. The assignment of
labels to devices is an auditable event.

A standard OpenVMS On-Line Disk Structure (ODS)-2 disk volume that has not been initialized by SEVMS
has a secrecy level of zero with no categories (system low). All of the files on the volume have the same
secrecy level and no categories. This facilitates using a OpenVMS volume as the system disk for SEVMS;
the volume does not need to be reinitialized after installing SEVMS. The SEVMS BACKUP utility can be
used to copy data from a OpenVMS disk volume to a labeled SEVMS disk volume.

Conclusion

SEVMS satisfies the BI Labels requirement.

8.4 Label Integrity

Requirement

Sensitivity labels shall accurately represent security levels of the specific subjects or objects with which they
are associated. When exported by the TCB, sensitivity labels shall accurately and unambiguously represent
the internal labels and shall be associated with the information being exported.

Applicable Features

All classification labels for subjects and objects are maintained by the SEVMS TCB and are protected from
unauthorized modification. The label associated with a subject may be specified by a user during login
processing, and may not be changed subsequently. If the user does not specify a label, the user’s default
label is used. In either case, the subject’s label must be within the authorized range specified for the user.
An administrator, using the AUTHORIZ utility, specifies the maximum, minimum, and default labels for
each user. When the DECnet software creates a unprivileged remote process to execute on behalf of a local
user, SEVMS assures that the classification of the remote process is the same as the classification of the
subject on the local node.

The classification label of all objects (except volumes) is initially set to the label of the creating subject. By
default, the range of labels for a volume is initialized to the authorized range of the user on whose behalf
the subject is executing. Alternatively, the subject may specify a range that is within the authorized range
of the user.

The classification label of all objects (except tape volumes) may be changed by an authorized subject using
the $SET.SECURITY system service. An untrusted subject may only raise the sensitivity of the object and
may only specify a single label value for both the maximum and minimum classification labels. To change
the classification labels of a tape volume, the volume must be reinitialized.

When a file is written to a disk, the classification label of the file is stored in the file header block of the file.
8.5 Exportation of Labeled Information

Requirement

The TCB shall designate each communication channel and I/O device as either single-level or multilevel. Any change in this designation shall be done manually and shall be auditable by the TCB. The TCB shall maintain and be able to audit any change in the current security level or levels associated with a communication channel or I/O device.

Applicable Features

SEVMS supports labels for terminals, workstation display devices, disk devices, tape devices, and printers; classification of other types of devices is not supported. All physically attached devices are owned by SYSTEM; therefore, devices may be labeled only by an administrator (in the SYSTEM category of users or having the SYSPRV privilege). The supported devices may be designated as either single-level or multilevel devices; the SEVMS TFM describes how to establish device labels during system installation and startup.

Devices that are strictly internal to the TCB (system mailboxes, cluster communication devices) are not labeled; they are protected from access by untrusted subjects using SEVMS discretionary access control mechanisms.

Disk, tape, and printer devices are supported by SEVMS as multilevel devices; the SECURITY privilege is required to designate a device as multilevel. The designation of other types of devices as multilevel is not supported in the evaluated configuration. The SEVMS TFM discusses the required constraints on designating devices as multilevel.

For terminal devices, SEVMS supports a login range, independent of the label designation of the terminal device. Therefore, terminal devices may be designated as single-level devices while allowing users to login over a range of classifications. When a user is active (logged in), the terminal label will be the same as the user’s process label; when no user is active, the terminal label will be as designated by the device label. The login range limits the user’s login label to be within the range specified.

Any changes to the label designation of a device and to the login range of a terminal are auditable (see page 163, “Auditing”).

Conclusion

SEVMS satisfies the Bl Label Integrity requirement.
Conclusion

SEVMS satisfies the Bl Exportation of Labeled Information requirement.

8.6 Exportation to Multilevel Devices

Requirement

When the TCB exports an object to a multilevel I/O device, the sensitivity label associated with that object shall also be exported and shall reside on the same physical medium as the exported information and shall be in the same form (i.e., machine-readable or human-readable form). When the TCB exports or imports an object over a multilevel communication channel, the protocol used on that channel shall provide for the unambiguous pairing between the sensitivity labels and the associated information that is sent or received.

Applicable Features

The multilevel devices supported by SEVMS are disks, tapes, and printers. For disk and tapes, the classification range of the volume mounted on the device must be within the range of the device; all device accesses must be within the volume range while the volume is mounted. The volume range is stored in SEVMS internal format in the volume headers of disk and tape volumes. The label associated with each file on the volume is stored in SEVMS internal format in the file headers on the volume. The SEVMS BACKUP utility preserves the original disk file labels within the backup file and restores them when the disk file is restored.

For printer devices, the SEVMS print symbiont provides human-readable labels that have been administratively defined to correlate to the internal label representations.

Conclusion

SEVMS satisfies the Bl Exportation to Multilevel Devices requirement.

8.7 Exportation to Single-Level Devices

Requirement

Single-level I/O devices and single-level communication channels are not required to maintain the sensitivity labels of the information they process. However, the TCB shall include a mechanism by which the TCB and an authorized user can reliably communicate to designate the single security level of information imported or exported via single-level communication channels or I/O devices.
Applicable Features

The labels of all single-level physically attached devices must be designated by an administrator (in the SYSTEM user category or having the SYSPRV privilege). SEVMS discretionary access control mechanisms are used to prevent untrusted subjects from changing device labels.

Data on an unlabeled tape volume may be imported by a subject with the IMPORT privilege. Before accessing the tape volume, the subject must designate a label for the data that is within the range for the tape device on which the volume is being mounted. Data may be exported to an unlabeled tape only by a subject with the DOWNGRADE or BYPASS privilege.

Conclusion

SEVMS satisfies the Bl Exportation to Single-Level Devices requirement.

8.8 Labeling Human-Readable Output

Requirement

The ADP system administrator shall be able to specify the printable label names associated with exported sensitivity labels. The TCB shall mark the beginning and end of all human-readable, paged, hardcopy output (e.g., line printer output) with human-readable sensitivity labels that properly \(^2\) represent the sensitivity of the output. The TCB shall, by default, mark the top and bottom of each page of human-readable, paged, hardcopy output (e.g., line printer output) with human-readable sensitivity labels that properly represent the overall sensitivity of the output or that properly represent the sensitivity of the information on the page. The TCB shall, by default and in an appropriate manner, mark other forms of human-readable output (e.g., maps, graphics) with human-readable sensitivity labels that properly represent the sensitivity of the output. Any override of these marking defaults shall be auditable by the TCB.

Applicable Features

SEVMS supports the association of a character string with each numeric classification level and each category designation. A system administrator uses the AUTHORIZE utility to define the correspondence. The character string representation, if defined, is used whenever the TCB displays a label in human-readable form; otherwise the numeric classification is used.

The SEVMS print symbiont provides a flexible mechanism for defining header and trailer pages and top and bottom page markings using administratively defined templates. The SEVMS default template prints the character string representation of the label on the header and trailer pages and at the top and bottom of each page (up to 29 characters for secrecy level and 199 characters for secrecy categories). Using a print template,

\(^2\)The hierarchical classification component in human-readable sensitivity labels shall be equal to the greatest hierarchical classification of any of the information in the output that the labels refer to; the non-hierarchical category component shall include all of the non-hierarchical categories of the information in the output the labels refer to, but no other non-hierarchical categories.
CHAPTER 8. EVALUATION AS A BI SYSTEM

a system administrator may define additional formats for use with specific printers, secrecy levels, secrecy categories, or printer widths. Only printers supported by the SEVMS print symbiont are in the evaluated configuration. For a VMCluster system, there is a single set of character string representations and a single set of print templates shared by all nodes in the evaluated configuration.

An untrusted subject may specify the /PASSALL option when printing a file to disable carriage control operations on the printer. The /PASSALL option also disables the output of the SEVMS page headers and footers. However by default, the SEVMS print symbiont disallows the use of the /PASSALL option. The print symbiont may be configured for specific printers to allow the use of the option and to audit its use.

Conclusion

SEVMS satisfies the BI Labeling Human-Readable Output requirement.

8.9 Subject Sensitivity Labels

Requirement

None.

Additional Requirement (B2)

NEW: The TCB shall immediately notify a terminal user of each change in the security level associated with that user during an interactive session. A terminal user shall be able to query the TCB as desired for a display of the subject’s complete sensitivity label.

Applicable Features

SEVMS provides a Digital Command Language (DCL) command to display the subject’s classification label and a system service that returns the subject’s classification to the calling program. A subject cannot change its sensitivity label during an interactive session.

Conclusion

SEVMS satisfies the B2 Subject Sensitivity Labels requirement.3

3 Although SEVMS satisfies this B2 functional requirement, it does not satisfy the assurance requirements above its rated level.
8.10 Device Labels

Requirement

None.

Additional Requirement (B2)

NEW: The TCB shall support the assignment of minimum and maximum security levels to all attached physical devices. These security labels shall be used by the TCB to enforce the constraints imposed by the physical environments in which the devices are located.

Applicable Features

SEVMS implements a range of labels to support multilevel disk, tape, and printer devices. In addition, a login range may be specified for terminal devices; a user may login at any label within the range.

Conclusion

SEVMS satisfies the B2 Device Labels requirement. 4

8.11 Mandatory Access Control

Requirement

The TCB shall enforce a mandatory access control policy over all subjects and storage objects under its control (e.g., processes, files, segments, devices). These subjects and objects shall be assigned sensitivity labels that are a combination of hierarchical classification levels and non-hierarchical categories, and the labels shall be used as the basis for mandatory access control decisions. The TCB shall be able to support two or more such security levels. The following requirements shall hold for all accesses between subjects and objects controlled by the TCB: A subject can read an object only if the hierarchical classification in the subject’s security level is greater than or equal to the hierarchical classification in the object’s security level and the non-hierarchical categories in the subject’s security level include all the non-hierarchical categories in the object’s security level. A subject can write an object only if the hierarchical classification in the subject’s security level is less than or equal to the hierarchical classification in the object’s security level and all the non-hierarchical categories in the subject’s security level are included in the non-hierarchical categories in the object’s security level. Identification and authentication data shall be used by the TCB to authenticate the user’s identity and to ensure that the security level and authorization of subjects external

4 Although SEVMS satisfies this B2 functional requirement, it does not satisfy the assurance requirements above its rated level.
to the TCB that may be created to act on the behalf of the individual user are dominated by the clearance and authorization of that user.

**Applicable Features**

In SEVMS, the storage objects protected by mandatory access controls are devices, files, volumes, shared logical name tables, global sections, common event flag clusters, queues, and resource domains. In the evaluated configuration, all objects have a single classification label except for disk devices, tape devices, printer devices, queues, and volumes, which may have an associated classification range (a minimum and maximum). The only physically attached devices supported by SEVMS are disk, tape, printer, terminal, and workstation display devices.

Each user has an associated minimum clearance label, maximum clearance label, and default clearance label maintained within the SEVMS security database by a system administrator. Each process is also assigned a classification label that is constrained by the TCB to be within the range specified in the security database for the user on whose behalf the process is operating. The classification label assigned to a process or object supports up to 256 hierarchical levels and 128 categories.

The SEVMS mandatory security policy enforces that a process (subject) has OBSERVE access to an object only if the label of the object dominates the (minimum) label of the object and a process has ALTER access to an object only if the (maximum) label of the object dominates the label of the process. One label dominates another if the hierarchical level is greater than or equal to the other and the category set includes all of the categories in the other set.

Three SEVMS privileges allow a subject to access an object in violation of the mandatory policy. The BYPASS privilege allows unconditional OBSERVE and ALTER access to all objects; the READALL privilege allows unconditional OBSERVE access to all objects; and the DOWNGRADE privilege allows ALTER access to objects whose labels do not dominate the subject’s label, as constrained by the SEVMS discretionary policy.

**Conclusion**

SEVMS satisfies the BI Mandatory Access Control requirement.

**8.12 Identification and Authentication**

**Requirement**

The TCB shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall maintain authentication data that includes information for verifying the identity of individual users (e.g., passwords) as well as information for determining the clearance and authorizations of individual users. This data shall be used by the TCB to authenticate the user’s identity and to ensure that the security level and authorizations of subjects external to the TCB that may be created to act on behalf of the individual user are dominated by the clearance and
authorization of that user. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user. The TCB shall be able to enforce individual accountability by providing the capability to uniquely identify each individual ADP system user. The TCB shall also provide the capability of associating this identity with all auditable actions taken by that individual.

Applicable Features

SEVMS provides support for passwords for all users; the applicable TFM describes how to assure that all login accounts require passwords to conform to the B1 evaluated configuration. The authentication data for a user is stored within the SEVMS security database (SYSUAF.DAT) and is protected from unauthorized access by SEVMS discretionary access control mechanisms. User passwords are encrypted within the database. However, when passwords are used with DECNET applications, the passwords are sent over the DECNET communication media unencrypted.\(^5\)

Each user is uniquely identified by username; the audit records generated by the TCB contain the username in all user activity records. Since all nodes in a VMScluster use the same security database, the identity of a user is the same on all nodes.

The user’s clearances (minimum, maximum, and default) are also stored within the SEVMS security database; SEVMS assures that each process created by a user has an associated classification label that is within the range specified by the security database.

Conclusion

SEVMS satisfies the B1 Identification and Authentication requirement.

8.13 Audit

Requirement

The TCB shall be able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data. The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into a user’s address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. The TCB shall also be able to audit any override of human-readable output markings. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user’s address space and for object deletion events the audit record shall include the name of the object and the object’s security level. The ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity and/or object security level.

\(^5\) In an SEVMS evaluated configuration, all of the TCB hardware must be physically protected.
Applicable Features

SEVMS provides the capability to audit certain specified security-related events and store them in an audit log and, optionally, an archive file. The log and archive files are protected such that only those possessing the appropriate access can manipulate their contents. The system is capable of auditing identification and authorization events such as logins and DECnet connection establishment. The system can also audit object-related functions such as open, read, write, and delete. Operator and system administrator activities can also be audited.

All audit records contain the date and time of an event, the type of event, and the associated username and process ID. For identification and authorization events, the source of the request (terminal ID or remote node) is included in the generated record. For object accesses, the audit record includes the type of object and its name.

The ANALYZE/AUDIT utility gives the user or administrator a powerful tool with which to select records from the audit log. Searches can be based on use of a particular privilege, on a user, or on an object security level.

In SEVMS, the audit record contains the object’s security level for mandatory access control events. SEVMS can also audit attempts to bypass labelling printed output.

Conclusion

SEVMS satisfies the BI Audit requirement.

8.14 System Architecture

Requirement

The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures). Resources controlled by the TCB may be a defined subset of the subjects and objects in the ADP system. The TCB shall maintain process isolation through the provision of distinct address spaces under its control. The TCB shall isolate the resources to be protected so that they are subject to the access control and auditing requirements.

Applicable Features

SEVMS includes features that protect it from external interference, provide address space isolation, and assure that access mediation and audit are performed for all protected resources.

SEVMS employs primarily memory management services to protect the TCB from interference. A processor supports four hierarchical hardware access modes for process execution. The modes are, from least privileged to most privileged: user, supervisor, executive, and kernel. For VAX processors, the current mode of execution is kept in the Processor Status Longword (PSL), and for Alpha the current mode of execution is
kept in the Processor Status (PS) register. Memory protection is based on hardware access modes; page protection codes are contained in the Page Table Entry (PTE) for each virtual page. For each hardware access mode, three types of access are defined: no access, read-only, and read-write. A page of memory accessible in a particular mode is also accessible in all of the more privileged modes. During address translation, the processor raises an access-control-violation fault if the current mode of execution is not authorized to access the page. TCB software establishes and maintains the page protection codes. Within the PTE an "owner" access mode is maintained by TCB software; changes to the protection code can only be made in an access mode at least as privileged as the owner access mode.

Access modes are used by TCB software to protect logical TCB data structures as well as pages of memory. For example, any logical names the TCB uses to resolve references to TCB data files will be defined by the TCB in a protected mode so they cannot be redefined by untrusted subjects. Record Management Services (RMS) uses access mode protection for resource locks to separate its resources from the resources managed by untrusted subjects. In general, logical data structures that can be created both by untrusted subjects and by the TCB will have associated access mode protection to isolate the TCB data structures.

SEVMS TCB data that is stored in files on disk is protected by the SEVMS discretionary access control mechanisms and is generally accessible only to subjects in the SYSTEM category. Other TCB data is protected by the use of SEVMS privileges; for example, a subject with the SYSNAM privilege is authorized to change the system directory of logical name tables.

SEVMS provides process address space isolation through its virtual memory management. Each process has a unique, virtual address space enforced through SEVMS paging mechanisms. The process virtual address space is divided into four logical regions. The system region in which the system executive resides is global to all processes; page protection is used to prevent unauthorized access. One other system region exists but is not used; the other two regions are process-unique. The two process-unique regions contain control structures and code and data, respectively.

The protected resources in SEVMS are disk, tape, printer, terminal, and VAXstation display devices, volumes, disk and tape files, and a number of SEVMS-specific objects that can be used for interprocess communication (global sections, resource domains, common event flag clusters, logical name tables, and queues). The SEVMS TCB controls all access to protected objects using system services and privileged images.

System services are implemented using the change mode instructions to transfer control from an untrusted program to the TCB. The change mode instructions transfer control to SEVMS change mode dispatchers (via the System Control Block (SCB)) and provide a numeric parameter. The execution access mode is raised to the specified mode; and the dispatcher, based on the parameter value, transfers control in the resulting access mode. The mode may only be increased; otherwise, the change mode instruction does not change the execution mode of the process. The Return from Exception or Interrupt (REI) instruction is used by the TCB to return control to the user program. SEVMS privileged images can be invoked by an untrusted subject to perform security-relevant operations. Each SEVMS privileged image performs specific operations on behalf of the user; when the image terminates, the privileges associated with the image are removed from the process. All security-relevant operations on protected resources are performed on behalf of an untrusted subject by SEVMS system services and privileged images.

SEVMS has a distributed architecture. Each node in a VMScluster configuration executes the TCB code independently. Local and cluster-visible resources are protected by the local TCB code. Access via DECnet software to non-cluster-visible resources are protected by the TCB code on the node where the resource is located. The security information upon which the policy is based is kept in a single security database,
CHAPTER 8. EVALUATION AS A BI SYSTEM

thereby enforcing a uniform security policy.

All operations on objects protected by the TCB are auditable by the TCB code that protects the resource. In addition to object creation, access, and deletion, events such as logins, logouts, login failures, changes to the security database, use of privileges, volume mounts and dismounts, and network management changes are auditable by the TCB. For a complete list of the types of auditable events, see page 163, “Auditing”.

Conclusion

SEVMS satisfies the BI System Architecture requirement.

8.15 System Integrity

Requirement

Hardware and/or software features shall be provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the TCB.

Applicable Features

VAX Systems

All VAX systems are supplied with a set of hardware diagnostics that are executed during power-on processing. These diagnostics check the basic hardware for proper operation and include tests of processor microcode, data path circuitry, and, in some processors, self-tests of attached devices. VAX hardware performs a number of consistency checks throughout normal operation, including processor checks and error correction functions.

In addition to the power-on diagnostics, Digital supports several diagnostic packages that are available through Digital Field Service. These packages include tests of virtual memory management, hardware access modes, page protection, privileged instructions, and the change mode instructions. They also include diagnostics for the attached hardware that can be configured with each processor.

Alpha Systems

All Alpha systems are supplied with a set of hardware tests that are executed during power-on processing. These tests check the basic hardware for proper operation and validate data path circuitry as well as system interconnects to memory and I/O devices. In addition to the power-on tests, Digital supports a test package, available through Digital Field Service, that tests the Alpha CPU. Alpha processors also provide extended console-selectable tests and exercisers for memory and I/O devices, and Alpha hardware performs a number of consistency checks throughout normal operation, including processor checks and error correction functions.

OpenVMS provides the User Environment Test Package (UETP) which is a system exerciser designed to stress the system by causing the system to perform all required functions. UETP can be configured to test any or all areas of the system. These include architectural features; peripherals; load testing the system;
network connections; memory; and cluster connections.

Many peripherals also have on-board tests that can be used to verify correct operation, thus allowing for additional verification of any attached device.

**Conclusion**

SEVMS satisfies the BI System Integrity requirement.

### 8.16 Security Testing

**Requirement**

The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. A team of individuals who thoroughly understand the specific implementation of the TCB shall subject its design documentation, source code, and object code to thorough analysis and testing. Their objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users. All discovered flaws shall be removed or neutralized and the TCB retested to demonstrate that they have been eliminated and that new flaws have not been introduced.

**Applicable Features**

Digital’s security test suite has evolved since SEVMS was initially evaluated. For the current RAMP Action, additional tests were added to the security test suite that was used for the RAMP of SEVMS VAX Version 6.1, and the security test suite was modified to run on both VAX and Alpha systems.

The security test suite was successfully executed at a Digital facility by Digital VSAs on a Mixed Architecture VMScluster (MAVC) system within a common security domain. The MAVC consisted of a VAX 8800, VAX 4000, AlphaServer 2100 and a DEC 7000.

**Conclusion**

SEVMS satisfies the BI Security Testing requirement.
8.17 Design Specification and Verification

Requirement

A formal or informal model of the security policy supported by the TCB shall be maintained over the life cycle of the ADP system and demonstrated to be consistent with its axioms.

Applicable Features

Digital has provided an informal security policy model that describes the SEVMS subjects and objects, and the discretionary and mandatory security properties enforced by the SEVMS TCB. For each protected object, the model shows how the TCB assures that the security properties are maintained. The model is described on page 179, “Informal Security Policy Model”.

Conclusion

SEVMS satisfies the BI Design Specification and Verification requirement.

8.18 Security Features User’s Guide

Requirement

A single summary, chapter, or manual in user documentation shall describe the protection mechanisms provided by the TCB, guidelines on their use, and how they interact with one another.

Applicable Features

VAX Systems

For SEVMS, the SFUG consists of Chapters 1-5 of the OpenVMS Guide to System Security[34] plus the VMS SES User’s Guide[48]. Chapters 1 and 2 of [34] describe the general nature of security problems and present an overview of the security policy. Chapter 3 covers user security on a single node and in a cluster environment. The intent of the chapter is to recommend good practices regarding the use of accounts and passwords. The topic of auditing is also discussed. Chapters 4 and 5 discuss the security objects and the associated protection mechanisms. Guidelines concerning management of the access control lists and default file access are also presented. The protection mechanisms are used to ensure that a requesting subject has the credentials (rights identifiers, hardware access modes, privileges) to access an object in the manner specified in the object’s ACL.

The SES User’s Guide[48] introduces the concept of Mandatory Access Control (MAC) and describes how it is implemented in SEVMS. MAC levels, as they apply to logins, classification of objects, tape and
backups, are discussed. The DCL commands and system services that handle MAC information are also described.

**Alpha Systems**

For SEVMS, the SFUG consists of Chapters 1-5 of the *OpenVMS Guide to System Security*\(^{[34]}\) plus the *SEVMS User’s Guide*\(^{[45]}\). Chapters 1 and 2 of \([34]\) describe the general nature of security problems and present an overview of the security policy. Chapter 3 covers user security on a single node and in a cluster environment. The intent of the chapter is to recommend good practices regarding the use of accounts and passwords. The topic of auditing is also discussed. Chapters 4 and 5 discuss the security objects and the associated protection mechanisms. Guidelines concerning management of the access control lists and default file access are also presented. The protection mechanisms are used to ensure that a requesting subject has the credentials (rights identifiers, hardware access modes, privileges) to access an object in the manner specified in the object’s ACL.

The *SEVMS User’s Guide*\(^{[45]}\) introduces the concept of MAC and describes how it is implemented in SEVMS. MAC levels, as they apply to logins, classification of objects, tape and backups, are discussed. The DCL commands and system services that handle MAC information are also described.

**Conclusion**

SEVMS satisfies the B1 Security Features User’s Guide requirement.

**8.19 Trusted Facility Manual**

**Requirement**

A manual addressed to the ADP system administrator shall present cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given. The manual shall describe the operator and administrator functions related to security, to include changing the security characteristics of a user. It shall provide guidelines on the consistent and effective use of the protection features of the system, how they interact, how to securely generate a new TCB, and facility procedures, warnings, and privileges that need to be controlled in order to operate the facility in a secure manner.

**Applicable Features**

**VAX Systems**

The TFM for SEVMS is composed of all of the components of the C2 trusted facility manual (refer to page 194, “Trusted Facility Manual”) plus the *VMS SES Security Manager’s Guide* \([47]\) and the *VMS SES Installation Guide*\(^{[46]}\). Chapters 1, 2, and 3 of the security manager’s guide provide an overview of the security features of SEVMS and their relationship to security features of OpenVMS. Chapter 4 describes
how to enter clearances for users and how to establish classifications for physical devices. Chapter 7 provides guidance on using the SEVMS audit mechanisms and Chapter 8 describes the use of the secure print facility.

Appendix C provides information on how to configure SEVMS to conform to the BI requirements of the evaluated configuration. The installation guide describes how to install the SEVMS system and set up an initial secure configuration.

Alpha Systems

The TFM for SEVMS is composed of all of the components of the C2 trusted facility manual (refer to page 194, “Trusted Facility Manual”) plus the SEVMS Security Manager’s Guide[44] and the SEVMS AXP Version 6.1 Installation Guide and Release Notes[43]. Chapters 1, 2, and 3 of the security manager’s guide provide an overview of the security features of SEVMS and their relationship to security features of OpenVMS. Chapter 4 describes how to enter clearances for users and how to establish classifications for physical devices. Chapter 6 provides guidance on using the SEVMS audit mechanisms and Chapter 7 describes the use of the secure print facility.

Appendix C provides information on how to configure SEVMS to conform to the BI requirements of the evaluated configuration. The installation guide describes how to install the SEVMS system and set up an initial secure configuration.

Conclusion

SEVMS satisfies the BI Trusted Facility Manual requirement.

8.20 Test Documentation

Requirement

The system developer shall provide to the evaluators a document that describes the test plan, test procedures that show how the security mechanisms were tested, and results of the security mechanisms’ functional testing.

Applicable Features

The Digital test plan, B1 Test Strategy for VMS and SEVMS[3], addresses the hardware and software requirements for executing the test suite, the overall testing approach, and the specific tests that are in the test suite. The basic approaches used are black box testing (functional testing of code without knowledge of what the code is doing), white box testing (functional testing of code with knowledge of how the code executes), and monolithic testing (testing all combinations of variables against a system). The monolithic and black box testing approaches are used to test Discretionary Access Control (DAC), and white box testing is used to test system services, DCL commands, utilities, object reuse, login, auditing, privilege use, and cluster operation.

The test suite is designed to execute a representative set of the possible access mediation scenarios to
sufficiently demonstrate that the security decision functions are implemented as intended. That is,

Either the system service or the associated DCL command that calls the service is used as the test interface. In some cases, both the command and the system service are tested. All operations involving access mediation are tested, including those operations requiring use of privilege. Combinations of subject and object access are used to test the access mediation mechanisms. Each of the possible access types for each object class are tested with subjects having different security characteristics. Testing of privileged operations is performed from the perspective of preserving the TCB boundary. That is, testing provides a measure of assurance that operations requiring privilege cannot be performed by subjects that do not possess the privilege. Other security mechanisms, such as the audit subsystem and identification and authentication features, are also covered by the test suite.

The approach to testing SEVMS extends beyond that described above. Digital has produced tests using the monolithic and black box testing approaches for MAC and human-readable labeling.

Digital provided evidence of the successful execution of the entire B1 security test suite in a VMScluster configuration.

Conclusion

SEVMS satisfies the B1 Test Documentation requirement.

8.21 Design Documentation

Requirement

Documentation shall be available that provides a description of the manufacturer’s philosophy of protection and an explanation of how this philosophy is translated into the TCB. If the TCB is composed of distinct modules, the interfaces between these modules shall be described. An informal or formal description of the security policy model enforced by the TCB shall be available and an explanation provided to show that it is sufficient to enforce the security policy. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model.

Applicable Features

VAX Systems

Digital has provided a Design Documentation Roadmap[8] that serves as an index to the security-relevant information in their extensive design documentation. This roadmap is built with an outline similar to the outline of this document. The roadmap points to sections of a number of secondary references. The list of secondary references to the design document are listed in the references to this report. These references describe the internal design of the system. Some modules of the system are documented very precisely while others are more high level. All of the TCB including all of the hardware components are documented to some degree.
CHAPTER 8. EVALUATION AS A B1 SYSTEM

The references of [8] are listed in the “References” appendix of this report. Several of the more important references are the VMS Internals and Data Structures: Version 5.2[80], VAXcluster Principles[74], VMS File System Internals[79], and VAX Architecture Reference Manual[93], [80], [74], and [79] cover the basic software architecture of OpenVMS and SEVMS, Version 6.1, with VAX or Alpha. [93] contains the basic hardware architecture. These references together form the basis of description of the TCB design.

Alpha Systems

Digital has provided an updated OpenVMS/SEVMS VAX and Alpha V6.1 Design Documentation Roadmap[8] that serves as an index to the security-relevant information in their extensive design documentation. This roadmap is built with an outline similar to the one of this document. The roadmap points to sections of a number of secondary references. The list of secondary references to the design document are listed in the references to this report. These references describe the internal design of the system. Some modules of the system are documented very precisely while others are more high level. All of the TCB including all of the hardware components are documented to some degree.

The references of [8] are listed in the “References” appendix of this report. Several of the more important references are OpenVMS AXP Internals and Data Structures, Version 1.5[72], Guidelines for VMScluster Configurations[13], VMScluster Systems for OpenVMS[40], VMS File System Internals[79], and Alpha Architecture Reference Manual[83]. [72], [13], [75], and [79] cover the basic software architecture of OpenVMS and SEVMS, Version 6.1, with VAX or Alpha. [83] contains the basic hardware architecture. These references together form the basis of description of the TCB design.

The informal security policy model [6] provides an explanation of how SEVMS enforces the security policy. For each protected object, the model identifies the TCB mechanisms (e.g., system services) that enforce the policy for that object.

Conclusion

SEVMS satisfies the B1 Design Documentation requirement.
Chapter 9
Evaluator’s Comments

OpenVMS and Security Enhanced VMS (SEVMS) are robust and flexible general purpose operating sys-
tems when run in evaluated configurations. Following are comments about OpenVMS and SEVMS. First,
several OpenVMS/SEVMS security features that are not addressed by the DoD Trusted Computer Sys-
tem Evaluation Criteria[2] (TCSEC) are discussed. This is followed by some general comments on the
evaluated systems.

9.1 Other OpenVMS/SEVMS Security Features

OpenVMS and SEVMS provide several security features that are not addressed by TCSEC requirements.
These include password management options, break-in detection mechanisms, extended discretionary access
controls, and support for application subsystems.

9.1.1 Password Management Options

OpenVMS/SEVMS provides a number of features that assist a system administrator to implement a site-
specific policy for managing passwords. A system password can be required for specific terminals, requiring
an additional authentication. If required, a user account can be set up to require two passwords instead
of one. A minimum password length and expiration date can be specified for each user account. The use
of a password generator can be required. In any case, OpenVMS/SEVMS checks a new password against
a dictionary of words; a system administrator may add site-specific words to the dictionary. Finally, an
administrator may choose to implement (but not within the evaluated configuration) site-specific password
filters and/or site-specific password encryption algorithms.

9.1.2 Break-in Detection

A system administrator can establish parameters for login attempts that will result in OpenVMS/SEVMS
taking evasive action when it detects a possible break-in attempt. A break-in database is maintained, in
addition to any audit records that might be generated.

9.1.3 Extended Discretionary Access Controls

The discretionary access control mechanisms of OpenVMS and SEVMS provide greater flexibility than a
simple group mechanism. A OpenVMS/SEVMS subject may “hold” many identifiers at the same time.
The system administrator controls the identifiers that are authorized for each user. Consequently, access
to objects can be controlled using identifier names that have meaning in particular application contexts. In addition, access control lists can specify that a subject must hold multiple identifiers to gain access. In this way, the owner of an object (or any subject authorized for CONTROL access to the object) can use identifiers to grant or withhold access at an appropriate granularity for the application.

9.1.4 Application Subsystems

The discretionary access control mechanisms of OpenVMS/SEVMS provide controls that can be used to authorize access to data files only when a user is executing an image authorized for access. An authorized subsystem administrator can designate image files as subsystem image files associated with a specific subsystem identifier. When the image is executed, the access rights list of a user is extended to include the identifier of the subsystem. Access control lists on the application data files can be set up to authorize access only to the subsystem identifier.

9.2 General Comments

This section presents several general comments about OpenVMS and SEVMS. The first six comments relate to both systems; the last four comments concern only SEVMS.

9.2.1 VMSCluster

The common VMScluster environment, where all nodes share a common user authorization database, provides a distributed computing environment where files and processing resources can be shared among the users on different cluster nodes. The applicable Trusted Facility Manual (TFM) for OpenVMS and SEVMS describes the environmental constraints needed for a VMScluster to meet the requirements for the evaluated configuration.

The OpenVMS/SEVMS distributed file system provides block-level I/O on cluster-visible devices, independent of the location of the physical device. Access control decisions for distributed files in the cluster are made in the same manner as for local files. The distributed lock manager provides concurrency controls to facilitate global sharing of files, records, and other cluster-wide resources. OpenVMS/SEVMS queue managers can be configured to manage the batch and print resources of the cluster based on the capabilities of each node. And the OpenVMS/SEVMS protection mechanisms can be used within the distributed environment to enforce a cluster-wide security policy (common security domain).

The flexibility of the VMScluster design allows a standalone system to be expanded into larger evaluated systems. A larger system can range from two processor nodes to a maximum of 128 processor nodes. The processor nodes can be any Virtual Address eXtension (VAX) or Alpha combination (i.e., all VAX; all Alpha; or combination of VAX and Alpha).
9.2.2 Internal Consistency Checks

OpenVMS/SEVMS is a robust system that does many internal consistency checks in normal operation. Data structures are typed, and the type field is checked for correctness throughout normal operations. The hardware also performs consistency checks. These checks include processor checks and error-correcting codes used in the memory architecture of VAX and Alpha systems.

9.2.3 Audit File Switching

The audit mechanism provided with OpenVMS/SEVMS monitors disk space on the volume containing the audit log file and provides warning messages to operator consoles. A system administrator may set thresholds where the audit server enters resource exhaustion mode, suspends system processes, and disables logins to slow down the generation of audit records. If the audit server’s internal buffers reach backlog thresholds, additional processes are suspended when they attempt to write an audit record in excess of a per-process limit.

To correct a resource exhaustion problem, a system administrator may either free disk space on the volume containing the audit log file or change the destination file to a different volume and create a new audit log file on that volume. Digital recommends that a new version of the audit log file be created daily; however, OpenVMS/SEVMS does not provide any automated switching of the audit log file from a full device to a secondary device during resource exhaustion.

9.2.4 Audit Alarms

The audit mechanism of OpenVMS/SEVMS provides a capability to immediately notify a system administrator if a specific type of auditable event occurs. By specifying an alarm, instead of or in addition to an audit, a system administrator can direct audit event messages to all terminals enabled to receive security event messages.

9.2.5 DECwindows

While DECwindows software is provided with both OpenVMS and SEVMS, it is not included in the evaluated configuration. The TFM describes how an administrator can disable DECwindows during system startup.

Without DECwindows, the usefulness of evaluated VAXstation configurations is limited. The graphics controllers on these VAX systems are not designed to support the monitor efficiently as a standard terminal monitor. Consequently, without DECwindows, the monitors are ineffective as user terminals. As an alternative, a standard Digital terminal can be attached to most VAXstation systems.

9.2.6 Trusted Path

OpenVMS/SEVMS provides a Secure Terminal Server that can be used to assure that the user communicates directly with the Trusted Computing Base (TCB) for login processing. The break key is used to initiate
the interaction with the TCB. This feature can only be used with standard terminal devices, not with workstation monitors.

A trusted path is provided when a card reader is used as the input device for a batch job running on a VAX processor. However, when DECnet authentication using password strings is employed, a trusted path to the remote cluster node is not provided.

9.2.7 Security Enhancement Services

SEVMS is marketed solely as part of a consulting package, Digital’s Security Enhancement Services (SES). The package includes a security review, security planning, a user orientation, a security manager orientation, and the installation of SEVMS.

9.2.8 Classified Mail

In SEVMS, the mail utility does not provide a true multilevel mail facility. The classification of a user’s mail directory determines the classification at which a user can read and send mail. The mail utility will “write-up” to put lower-level mail messages into a user’s mail directory. However, such lower-level mail messages are labeled at the level of the mail directory and can only be read when the user is logged in at that classification. Consequently, reading and sending mail can only occur at a single level—the classification of the user’s mail directory.

9.2.9 Integrity Policy

SEVMS includes some functionality for mandatory integrity. A mandatory integrity policy could provide additional mechanisms for protecting both TCB and user data from Trojan horse and virus attacks. The integrity policy in SEVMS was not included in the informal security policy model and is not discussed in depth in the SEVMS user documentation. The SEVMS TCB does not use the mandatory integrity mechanism to provide any additional protection for internal TCB data.

The evaluation team attempted to use the integrity mechanism during team testing and determined that it would be difficult to use. All of the SEVMS software is by default labeled at “system low” integrity instead of “system high” integrity. Therefore, a subject with high integrity cannot execute SEVMS programs without violating the integrity policy.

9.2.10 Classification of Files and Directories

In SEVMS, an untrusted subject may upgrade the classification of a file or directory. This can result in problems if a user does not adhere to the guidance in the SEVMS Security Features User’s Guide (SFUG). For example, an untrusted subject can raise the label of a directory that already contains lower level files. In this case, the lower level files can be accessed via a pathname only when the subject’s classification dominates the directory classification. They can still be accessed directly by file identifier at the lower classification.

---

1 Card readers are not available on Alpha systems.
However, to delete a lower level file and its directory entry, a subject would need the DOWNGRADE
privilege.
Part IV

Appendixes
Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha
Appendix A

Evaluated Hardware List (VAX and Alpha)

This appendix identifies the hardware included in the OpenVMS/SEVMS evaluated configuration. The hardware must be configured in accordance with Digital's requirements for system configuration, since Digital does not support all possible hardware configurations. Information about Digital supported configurations can be obtained by contacting a Digital sales office. For any evaluated configuration, the processor(s) and all associated hardware must be listed within this appendix, and the configuration must be supported by Digital. The first section of this appendix identifies the processors included in the evaluation. The second section identifies this same collection of processors, but grouped by system bus. The other sections of this appendix identify the I/O devices and other hardware included in the evaluated configuration.

A.1 Processors

VAX-11/730
VAX-11/750
VAX-11/780
VAX-11/785
MicroVAX II
VAXstation II/GPX\(^1\)
VAXstation II/QVSS\(^2\)
MicroVAX 2000
VAXstation 2000, VAXstation 2000/GPX, VAXstation 2000/MFB\(^3\)
AlphaServer 2100 Models 4/200, 4/275
MicroVAX 3100, Models 10, 10E, 20, 20E, 30, 40, 80, 90, 95
VAXserver 3100, Models 10, 10E, 20, 20E
VAXstation 3100, Models 30, 38, 40, 48, 76
VAXstation 3100/GPX, Models 38, 48, 76
VAXstation 3100/SPX\(^4\) Models 38, 48, 76
MicroVAX 3200
VAXserver 3200
VAXstation 3200
MicroVAX 3300
VAXserver 3300
MicroVAX 3400
VAXserver 3400
MicroVAX 3500
VAXserver 3500
VAXstation 3500

\(^1\) Graphics Processing Accelerator (GPX)
\(^2\) Q-Bus Video Sub System (QVSS)
\(^3\) Monochrome Frame Buffer (MFB)
\(^4\) 2D Scanline Processor Accelerator Graphics System (SPX)
VAXstation 3520
VAXstation 3540
MicroVAX 3600
VAXserver 3600
MicroVAX 3800
VAXserver 3800
MicroVAX 3900
VAXserver 3900
VAX 4000, Models 50, 100A, 500A, 600A, 700A
VAX 4000, Models 100, 105, 200, 300, 500, 600
VAXserver 4000, Models 200, 300, 400, 500, 600
VAXstation 4000, Models 60, 90
VAXstation 4000 VLC
VAX 6000 Series, Models 210, 220, 230, 240
VAX 6000 Series, Models 310, 320, 330, 340, 360
VAX 6000 Series, Models 410, 420, 430, 440, 450, 460
VAX 6000 Series, Models 510, 520, 530, 540, 550, 560
VAX 6000 Series, Models 610, 620, 630, 640
VAXserver 6000, Models 210, 220, 310, 320, 410, 420, 510, 520
DEC 7000, Models 610, 620, 630, 640, 650, 660
VAX 7000, Models 610, 620, 630, 640, 650, 660
DEC 7000, Models 710, 720, 730, 740, 750, 760
VAX 7000, Models 710, 720, 730, 740, 750, 760
VAX 8200, VAX 8250
VAX 8300, VAX 8350
VAX 8530, VAX 8550
VAX 8600, VAX 8650
VAX 8700
VAX 8800, VAX 8810, VAX 8820, VAX 8830, VAX 8840
VAXserver 8200, VAXserver 8250, VAXserver 8300, VAXserver 8350
VAXserver 8530, VAXserver 8550
VAXserver 8600, VAXserver 8650
VAXserver 8700
VAXserver 8800, VAXserver 8810, VAXserver 8820, VAXserver 8830, VAXserver 8840
VAX 9000, Models 110, 110VP, 210, 210VP, 310, 310VP
VAX 9000, Models 320, 320VP, 330, 330VP, 340, 340VP
VAX 9000, Models 410, 410VP, 420, 420VP, 430, 430VP
VAX 9000, Models 440, 440VP
DEC 10000, Models 610, 620, 630, 640, 650, 660
VAX 10000, Models 610, 620, 630, 640, 650, 660
DEC 10000, Models 710, 720, 730, 740, 750, 760
VAX 10000, Models 710, 720, 730, 740, 750, 760
Norden MIL VAX II
Raytheon MVCf 860
Raytheon MVCf 860

---

5 Vector Processor (VP)
6 A militarized version of the VAX-11/750.
7 A militarized version of the VAX 6000 Model 200 series.
8 A militarized version of the VAX 6000 Model 600 series.

FINAL: 24 October 1996
A.2 Processors - Grouped by System Bus

Q-Bus System Bus
MicroVAX II
VAXstation II/QVSS, VAXstation II/GPX
MicroVAX 3200
VAXserver 3200
VAXstation 3200
MicroVAX 3300
VAXserver 3300
MicroVAX 3400
VAXserver 3400
MicroVAX 3500
VAXserver 3500
VAXstation 3500
VAXstation 3520, VAXstation 3540
MicroVAX 3600
VAXserver 3600
MicroVAX 3800
VAXserver 3800
MicroVAX 3900
VAXserver 3900
VAX 4000, Models 50, 100A, 500A, 600A, and 700A
VAX 4000, Models 100, 105, 200, 300, 500, 600
VAXserver 4000, Models 200, 300, 400, 500, 600

NMI System Bus
VAX 8530, VAX 8550
VAX 8700
VAX 8800, VAX 8810, VAX 8820, VAX 8830, VAX 8840
VAXserver 8700
VAXserver 8800, VAXserver 8810, VAXserver 8820, VAXserver 8830, VAXserver 8840

XMI System Bus
VAX 6000 Series, Models 210, 220, 230, 240
Raytheon MVC 860
VAX 6000 Series, Models 310, 320, 330, 340, 360
VAX 6000 Series, Models 410, 420, 430, 440, 450, 460
VAX 6000 Series, Models 510, 520, 530, 540, 550, 560
VAX 6000 Series, Models 610, 620, 630, 640
Raytheon MVC 866
VAXserver 6000, Models 210, 220, 310, 320, 410, 420, 510, 520
VAX 8530, VAX 8550
VAX 8700
VAXserver 8800, VAXserver 8810, VAXserver 8820, VAXserver 8830, VAXserver 8840

Laser System Bus
DEC 7000, Models 610, 620, 630, 640, 650, 660
VAX 7000, Models 610, 620, 630, 640, 650, 660
Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha

APPENDIX A. EVALUATED HARDWARE LIST (VAX AND ALPHA)

DEC 7000, Models 710, 720, 730, 740, 750, 760
VAX 7000, Models 710, 720, 730, 740, 750, 760
DEC 10000, Models 610, 620, 630, 640, 650, 660
VAX 10000, Models 610, 620, 630, 640, 650, 660
DEC 10000, Models 710, 720, 730, 740, 750, 760
VAX 10000, Models 710, 720, 730, 740, 750, 760

VAXBI System Bus
VAX 8200, VAX 8250
VAX 8300, VAX 8350
VAXserver 8200, VAXserver 8250
VAXserver 8300, VAXserver 8350

SBI System Bus
VAX-11/780
VAX-11/785
VAX 8600, VAX 8650
VAXserver 8600, VAXserver 8650

CMI System Bus
VAX-11/750
Norden MIL VAX II

Special Internal System Bus
VAX 11/730
MicroVAX 2000
VAXstation 2000
VAXstation 2000/GPX
VAXstation 2000/MFB
VAXft 3000, Models 110, 310, 410, 610, 612
MicroVAX 3100, Models 10, 10E, 20, 20E, 30, 40, 80, 90, 95
VAXserver 3100, Models 10, 10E, 20, 20E
VAXstation 3100, Models 30, 38, 40, 48, 76
VAXstation 3100/GPX, Models 38, 48, 76
VAXstation 3100/SPX, Models 38, 48, 76
VAXstation 4000, Models 60, 90
VAXstation 4000 VLC
VAX 9000, Models 110, 110VP
VAX 9000, Models 210, 210VP
VAX 9000, Models 310, 310VP, 320, 320VP, 330, 330VP, 340, 340VP
VAX 9000, Models 410, 410VP, 420, 420VP, 430, 430VP
VAX 9000, Models 440, 440VP

COBRA System Bus
AlphaServer 2100 Models 4/200, 4/275

FINAL: 24 October 1996
A.3 I/O Busses for AlphaServer 2100 Models 4/200, 4/275

PCI Bus Generic
EISA Bus Generic

A.4 I/O Bus Adapters and Controllers

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMB32-LP</td>
<td>VAXBI DMA parallel high speed line printer controller.</td>
</tr>
<tr>
<td>DRB32</td>
<td>High speed general purpose parallel interface for VAXBI based systems.</td>
</tr>
<tr>
<td>DMF32-LP</td>
<td>Synchronous/Asynchronous communication option with printer port for use on UNIBUS based systems.</td>
</tr>
<tr>
<td>HSC40</td>
<td>Hierarchical Storage Controller for MSCP disks and TMSCP tapes.</td>
</tr>
<tr>
<td>HSC50</td>
<td>Hierarchical Storage Controller for MSCP disks and TMSCP tapes.</td>
</tr>
<tr>
<td>HSC60</td>
<td>Hierarchical Storage Controller for MSCP disks and TMSCP tapes.</td>
</tr>
<tr>
<td>HSC65</td>
<td>Hierarchical Storage Controller for MSCP disks and TMSCP tapes.</td>
</tr>
<tr>
<td>HSC70</td>
<td>Hierarchical Storage Controller for MSCP disks and TMSCP tapes.</td>
</tr>
<tr>
<td>HSC90</td>
<td>Hierarchical Storage Controller for MSCP disks and TMSCP tapes.</td>
</tr>
<tr>
<td>HSC95</td>
<td>Hierarchical Storage Controller for MSCP disks and TMSCP tapes.</td>
</tr>
<tr>
<td>HSD20</td>
<td>DSSI RAID disk subsystem.</td>
</tr>
<tr>
<td>HSD30</td>
<td>DSSI RAID disk subsystem.</td>
</tr>
<tr>
<td>HSJ40</td>
<td>CI RAID disk subsystem.</td>
</tr>
<tr>
<td>HSZ40</td>
<td>SCSI RAID disk subsystem.</td>
</tr>
<tr>
<td>IDC</td>
<td>Integrated Disk Controller for VAX-11/725 and VAX-11/730 systems.</td>
</tr>
<tr>
<td>IDTC</td>
<td>Integral Disk and Tape Controller for VAX 8600 and VAX 8650 systems.</td>
</tr>
<tr>
<td>KDA50</td>
<td>Q-bus MSCP disk controller.</td>
</tr>
<tr>
<td>KDB50</td>
<td>VAXBI MSCP disk controller.</td>
</tr>
<tr>
<td>KDM70</td>
<td>A intelligent MSCP/TMSCP mass-storage controller that supports RA Series Disks and Storage Arrays, TA Series Tape, and ESE20.</td>
</tr>
<tr>
<td>KFMSA</td>
<td>XMI to DSSI bus storage adapter. (VAX only)</td>
</tr>
<tr>
<td>KFMSB</td>
<td>XMI to DSSI bus storage adapter. (Alpha only)</td>
</tr>
<tr>
<td>KFQSA</td>
<td>Q-bus DSSI bus storage adapter.</td>
</tr>
<tr>
<td>KLESI</td>
<td>Q-bus, UNIBUS, and VAXBI tape controller.</td>
</tr>
<tr>
<td>KRRQ50</td>
<td>Q-bus controller for the RRD40/RRD50 compact disk reader.</td>
</tr>
<tr>
<td>KZQSA</td>
<td>Q-bus to SCSI bus adapter.</td>
</tr>
<tr>
<td>KFD DA</td>
<td>VAX 4000 model 100 DSSI bus adapter.</td>
</tr>
<tr>
<td>KFD DB</td>
<td>VAX 4000 model 500/600/700 DSSI bus adapter.</td>
</tr>
<tr>
<td>LP11</td>
<td>UNIBUS parallel high-speed line printer controller for LPxx printers.</td>
</tr>
<tr>
<td>LPV11</td>
<td>Q-bus parallel high-speed line printer controller.</td>
</tr>
</tbody>
</table>
## APPENDIX A. EVALUATED HARDWARE LIST (VAX AND ALPHA)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RK711</td>
<td>UNIBUS disk controller for RK07 disk drives.</td>
</tr>
<tr>
<td>RL211</td>
<td>UNIBUS disk controller for the RL02 disk drive.</td>
</tr>
<tr>
<td>RQDX8</td>
<td>Q-bus disk controller for MicroVAX and VAXstation systems.</td>
</tr>
<tr>
<td>RX211</td>
<td>UNIBUS diskette controller.</td>
</tr>
<tr>
<td>RUX50</td>
<td>UNIBUS diskette controller.</td>
</tr>
<tr>
<td>TM03</td>
<td>MASSBUS tape controller.</td>
</tr>
<tr>
<td>TM32</td>
<td>BI-bus 9-track tape controller only with large record support.</td>
</tr>
<tr>
<td>TM78</td>
<td>MASSBUS tape controller.</td>
</tr>
<tr>
<td>TQK50</td>
<td>Q-bus tape controller.</td>
</tr>
<tr>
<td>TQK70</td>
<td>Q-bus tape controller.</td>
</tr>
<tr>
<td>TS11</td>
<td>UNIBUS tape controller.</td>
</tr>
<tr>
<td>TBK50</td>
<td>BI-bus tape controller.</td>
</tr>
<tr>
<td>TBK70</td>
<td>BI-bus tape controller.</td>
</tr>
<tr>
<td>TSV05</td>
<td>Q-bus tape controller for the TS05.</td>
</tr>
<tr>
<td>TSU05</td>
<td>Unibus tape controller for the TS05.</td>
</tr>
<tr>
<td>TUK50</td>
<td>UNIBUS tape controller.</td>
</tr>
<tr>
<td>UDA50</td>
<td>UNIBUS MSCP disk controller.</td>
</tr>
<tr>
<td>CXA16</td>
<td>16 line serial terminal multiplexer.</td>
</tr>
<tr>
<td>CXB16</td>
<td>16 line serial terminal multiplexer.</td>
</tr>
<tr>
<td>CYX08</td>
<td>8 line serial terminal multiplexer.</td>
</tr>
<tr>
<td>DHB32</td>
<td>16 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DHF11</td>
<td>32 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DHT32</td>
<td>8 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DHQ11</td>
<td>8 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DHU11</td>
<td>16 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DHV11</td>
<td>8 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DMB32</td>
<td>8 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DMF32</td>
<td>8 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DMZ32</td>
<td>24 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DSH32</td>
<td>1 line synchronous and 8 line asynchronous communications controller.</td>
</tr>
<tr>
<td>DZ11</td>
<td>8 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DZ32</td>
<td>8 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DZQ11</td>
<td>4 line asynchronous terminal controller.</td>
</tr>
<tr>
<td>DZV11</td>
<td>4 line asynchronous terminal controller.</td>
</tr>
</tbody>
</table>
DEFAA Futurebus to FDDI controller.
KZESC EISA to SCSI controller.
KZMSA XMI to SCSI controller.
KZPSC PCI to SCSI controller.
KZPAA PCI to SCSI controller.
DEUNA Ethernet to UNIBUS controller.
DELUA Ethernet to UNIBUS controller.
DEBNA Ethernet to VAXBI communication controller.
DEBNI Ethernet to VAXBI communication controller.
DEMFA The DEMFA is a high performance network adapter that connects XMI systems to both Ethernet and IEEE FDDI.
DEMNA The DEMNA is a high performance network adapter that connects XMI systems to both the Ethernet and IEEE 802.3 local area networks.
DESVG Ethernet controller interface.
DEQNA Ethernet controller to Q-bus.
DELQA Ethernet controller to Q-bus.
DESQA Ethernet controller to Q-bus.
DEFQA FDDI to Q-bus controller.
DEFTA FDDI to Turbochannel controller.
DEFZA FDDI to Turbochannel controller.
KFE52 DSSI/Ethernet Adapter.

CI750 CI Adapter for VAX-11/750 systems.
CI780 CI Adapter for VAX-11/780, VAX-11/785, VAX 8600, and VAX 8650 systems.
CIBC1 CI Adapter for VAXBI systems.
CIBCA-AA Native CI Adapter for VAXBI systems.
CIBCA-BA Native CI Adapter for VAXBI systems.
CIQCD-AA Native CI Adapter for VAX 9xxx XMI systems.
CIQCD-AB Native CI Adapter for VAX 6xxx XMI systems.
CIQCD-AC Native CI Adapter for VAX 7xxx/10xxx XMI systems.
DWBUA XMI to UNIBUS adapter.
DWMDA XMI to Bl adapter.
DWVMA XMI to VME adapter.
DW750 UNIBUS adapter.
DW780 UNIBUS adapter.
RH750 MASSBUS adapter.
RH780 MASSBUS controller.
A.5 Disk Devices

EF51R 107 MB solid state disk drive.
EF52R 205 MB solid state disk drive.
EF53 267 MB solid state disk drive.
ESE-20 120 MB solid state disk drive.
ESE-52 120 MB solid state disk drive.
ESE-56 600 MB solid state disk drive.
ESE-58 960 MB solid state disk drive.
EZ51R 100 MB solid state disk drive.
EZ54R 467 MB solid state disk drive.
EZ58R 855 MB solid state disk drive.
RA60 205 MB removable disk drive.
RA70 280 MB fixed disk drive.
RA71 700 MB fixed disk drive.
RA72 1 GB fixed disk drive.
RA73 2 GB fixed disk drive.
RA80 128 MB fixed disk drive.
RA81 456 MB fixed disk drive.
RA82 622 MB fixed disk drive.
RA90 1.2 GB fixed disk drive.
RA92 1.5 GB fixed disk drive.
RC25* 2 disks each 26 MB (1 fixed and 1 removable) disk drive with shared spindle.
RD51* 10 MB fixed disk drive.
RD52* 31 MB fixed disk drive.
RD53* 71 MB fixed disk drive.
RD54 159 MB fixed disk drive.
RF30 150 MB fixed disk drive.
RF30-RA 150 MB removable disk drive.
RF31F 200 MB fixed disk drive.
RF31-JA 381 MB shock-mounted removable disk drive.
RF31-KA 381 MB fixed disk drive.
RF31T 381 MB fixed disk drive.
RF35 800 MB fixed disk drive.
RF36 1.6 GB fixed disk drive.
RF71 400 MB disk drive.
RF71-RA 400 MB removable disk drive.
RF72 1 GB removable disk drive.
RF72-RA 1 GB removable disk drive.
RF73 2 GB fixed disk drive.
RF74 3.5 GB fixed disk drive.
### DISK DEVICES

<table>
<thead>
<tr>
<th>Device Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RK06*</td>
<td>14 MB removable disk drive.</td>
</tr>
<tr>
<td>RK07*</td>
<td>28 MB removable disk drive.</td>
</tr>
<tr>
<td>RL02*</td>
<td>10 MB removable disk drive.</td>
</tr>
<tr>
<td>RM03</td>
<td>67 MB removable disk drive.</td>
</tr>
<tr>
<td>RM05</td>
<td>256 MB removable disk drive.</td>
</tr>
<tr>
<td>RM80</td>
<td>124 MB fixed disk drive.</td>
</tr>
<tr>
<td>R80</td>
<td>124 MB fixed disk drive for VAX-11/725 and VAX-11/730.</td>
</tr>
<tr>
<td>RRD40*</td>
<td>600 MB read-only optical disk drive.</td>
</tr>
<tr>
<td>RRD42</td>
<td>600 MB read-only optical disk drive.</td>
</tr>
<tr>
<td>RRD43</td>
<td>600 MB read-only optical disk drive.</td>
</tr>
<tr>
<td>RRD50*</td>
<td>600 MB read-only optical disk drive.</td>
</tr>
<tr>
<td>RP05</td>
<td>88 MB removable disk drive.</td>
</tr>
<tr>
<td>RP06</td>
<td>176 MB removable disk drive.</td>
</tr>
<tr>
<td>RP07</td>
<td>516 MB fixed disk drive.</td>
</tr>
<tr>
<td>RVZ01</td>
<td>594 MB optical removable disk drive.</td>
</tr>
<tr>
<td>RX.02*</td>
<td>512 KB diskette drive.</td>
</tr>
<tr>
<td>RX.23</td>
<td>1.47 MB diskette drive.</td>
</tr>
<tr>
<td>RX.26</td>
<td>2.8 MB diskette drive.</td>
</tr>
<tr>
<td>RX.33*</td>
<td>1.2 MB diskette drive.</td>
</tr>
<tr>
<td>RX.50*</td>
<td>400 KB diskette drive.</td>
</tr>
<tr>
<td>RV20*</td>
<td>2 GB Write Once Read Many optical disk drive.</td>
</tr>
<tr>
<td>RV64*</td>
<td>2 GB Write Once Read Many optical disk sub-system.</td>
</tr>
<tr>
<td>RZ22*</td>
<td>52 MB fixed disk drive.</td>
</tr>
<tr>
<td>RZ23</td>
<td>104 MB fixed disk drive.</td>
</tr>
<tr>
<td>RZ23L</td>
<td>121 MB fixed disk drive.</td>
</tr>
<tr>
<td>RZ24</td>
<td>200 MB fixed disk drive.</td>
</tr>
<tr>
<td>RZ24L</td>
<td>240 MB fixed disk drive.</td>
</tr>
<tr>
<td>RZ25</td>
<td>425 MB fixed disk drive.</td>
</tr>
<tr>
<td>RZ25L</td>
<td>500 MB fixed disk drive.</td>
</tr>
<tr>
<td>RZ26</td>
<td>1.0 GB fixed disk drive.</td>
</tr>
<tr>
<td>RZ26B</td>
<td>1.0 GB fixed disk drive.</td>
</tr>
<tr>
<td>RZ26L</td>
<td>1.0 GB fixed disk drive.</td>
</tr>
<tr>
<td>RZ28</td>
<td>2.1 GB fixed disk drive.</td>
</tr>
<tr>
<td>RZ28B</td>
<td>2.1 GB fixed disk drive.</td>
</tr>
<tr>
<td>RZ35</td>
<td>852 MB fixed disk drive.</td>
</tr>
<tr>
<td>RZ35L</td>
<td>332 MB fixed disk drive.</td>
</tr>
<tr>
<td>RZ36</td>
<td>665 MB fixed disk drive.</td>
</tr>
<tr>
<td>RZ37</td>
<td>1 GB fixed disk drive.</td>
</tr>
<tr>
<td>RZ38</td>
<td>1.35 GB fixed disk drive.</td>
</tr>
<tr>
<td>RZ73</td>
<td>2 GB fixed disk drive.</td>
</tr>
<tr>
<td>RZ74</td>
<td>3.5 GB fixed disk drive.</td>
</tr>
</tbody>
</table>

* Device cannot be used as a VMS system disk.
Appendix A. Evaluated Hardware List (VAX and Alpha)

A.6 Tape Devices

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA78</td>
<td>STI TU78.</td>
</tr>
<tr>
<td>TA79</td>
<td>STI TU79.</td>
</tr>
<tr>
<td>TA81</td>
<td>STI TU81.</td>
</tr>
<tr>
<td>TA90</td>
<td>1.2 GB tape cartridge subsystem.</td>
</tr>
<tr>
<td>TA90E</td>
<td>1.2 GB tape cartridge subsystem.</td>
</tr>
<tr>
<td>TA91</td>
<td>High performance tape drive.</td>
</tr>
<tr>
<td>TE16</td>
<td>9-track magnetic tape drive.</td>
</tr>
<tr>
<td>TF70</td>
<td>290 MB TK70 tape cartridge drive.</td>
</tr>
<tr>
<td>TF85</td>
<td>2.6 GB tape drive.</td>
</tr>
<tr>
<td>TF857</td>
<td>18.2 GB tape cartridge loader.</td>
</tr>
<tr>
<td>TK50</td>
<td>95 MB, 5 1/4 inch streaming tape cartridge.</td>
</tr>
<tr>
<td>TK70</td>
<td>296 MB, 5 1/4 inch streaming tape cartridge.</td>
</tr>
<tr>
<td>TKZ09</td>
<td>5.25 inch, 5 GB, 8mm tape drive.</td>
</tr>
<tr>
<td>TKZ60</td>
<td>3480 compatible tape drive.</td>
</tr>
<tr>
<td>TLZ04</td>
<td>4 GB, 3.5 inch, 4 mm tape drive.</td>
</tr>
<tr>
<td>TLZ06</td>
<td>4 GB, 3.5 inch, 4 mm DAT tape drive.</td>
</tr>
<tr>
<td>TLZ6L</td>
<td>4 GB, 3.5 inch, 4mm DAT tape loader.</td>
</tr>
<tr>
<td>TSZ05</td>
<td>1600 BPI, 9-track tape drive.</td>
</tr>
<tr>
<td>TSZ07</td>
<td>1600/6250 BPI tape drive.</td>
</tr>
<tr>
<td>TS11</td>
<td>9-track magnetic tape drive.</td>
</tr>
<tr>
<td>TU77</td>
<td>9-track magnetic tape drive.</td>
</tr>
<tr>
<td>TU78</td>
<td>9-track magnetic tape drive.</td>
</tr>
<tr>
<td>TU80</td>
<td>9-track magnetic tape drive.</td>
</tr>
<tr>
<td>TU81</td>
<td>9-track magnetic tape drive.</td>
</tr>
<tr>
<td>TU81-Plus</td>
<td>Streaming 9-track magnetic tape drive.</td>
</tr>
<tr>
<td>TZ30</td>
<td>95 MB, 5 1/4 inch, half-height, tape drive.</td>
</tr>
<tr>
<td>TZ85</td>
<td>2.6 GB, 5.25 inch, tape drive.</td>
</tr>
<tr>
<td>TZ857</td>
<td>18.2 GB tape cartridge loader.</td>
</tr>
<tr>
<td>TZ867</td>
<td>42 GB tape cartridge loader.</td>
</tr>
<tr>
<td>TZ87</td>
<td>5.25 inch, tape drive.</td>
</tr>
<tr>
<td>TZ877</td>
<td>140 GB tape cartridge loader.</td>
</tr>
<tr>
<td>TZEK10</td>
<td>320/525 MB QIC (quarter inch cartridge) tape drive.</td>
</tr>
</tbody>
</table>

A.7 Terminals

Miscellaneous
- VT05
- VT5X
- VT52
- VT55
- VT80
- VK100

VT100 Series
- VT100
- VT101
VT102
VT105
VT125
VT131
VT132
VT173

**VT200 Series**
VT220
VT225
VT240
VT241
VT278
VT282
VT284
VT286

**VT300 Series**
VT320
VT330
VT340
VT382
VT383

**VT400 Series**
VT420 (C2 configuration only)

**VT500 Series**
VT510 (C2 configuration only)
VT520 (C2 configuration only)
VT525 (C2 configuration only)

**LA Series**
LA12
LA24
LA34
LA36
LA38
LA80
LA84
LA100
LA120
LA210
A.8 Printers

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA100</td>
<td>Letterwriter</td>
</tr>
<tr>
<td>LA12</td>
<td>DECwriter Correspondant</td>
</tr>
<tr>
<td>LA120</td>
<td>LA120 DECwriter and DECprinter III</td>
</tr>
<tr>
<td>LA210</td>
<td>Letterprinter</td>
</tr>
<tr>
<td>LA310</td>
<td>Multiprinter</td>
</tr>
<tr>
<td>LA324</td>
<td>MultiPrinter</td>
</tr>
<tr>
<td>LA50</td>
<td>Personal Printer</td>
</tr>
<tr>
<td>LA70</td>
<td>Personal Printer</td>
</tr>
<tr>
<td>LA75</td>
<td>Companion Printer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCP01</td>
<td>Color Printer</td>
</tr>
<tr>
<td>LG01</td>
<td>Dot-matrix Lineprinter</td>
</tr>
<tr>
<td>LG02</td>
<td>Dot-matrix Lineprinter</td>
</tr>
<tr>
<td>LG06</td>
<td>Dot-matrix Lineprinter</td>
</tr>
<tr>
<td>LG12</td>
<td>Dot-matrix Lineprinter</td>
</tr>
<tr>
<td>LG31</td>
<td>Dot-matrix Lineprinter</td>
</tr>
<tr>
<td>LJ250</td>
<td>Companion Lineprinter</td>
</tr>
<tr>
<td>LJ252</td>
<td></td>
</tr>
<tr>
<td>LN01</td>
<td>Laser Page Printer</td>
</tr>
<tr>
<td>LN01S</td>
<td>Laser Printer</td>
</tr>
<tr>
<td>LN01B</td>
<td>Laser Printer</td>
</tr>
<tr>
<td>LN03</td>
<td>Laser Printer</td>
</tr>
<tr>
<td>LN03S</td>
<td>PLUS Laser Printer</td>
</tr>
<tr>
<td>LN05</td>
<td>DECprinter 2100</td>
</tr>
<tr>
<td>LN06</td>
<td>DECprinter 2200</td>
</tr>
<tr>
<td>LN07</td>
<td>DECprinter 1100</td>
</tr>
<tr>
<td>LN08</td>
<td>DECprinter 3200</td>
</tr>
<tr>
<td>LN09</td>
<td>DECprinter 5100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LQP02</td>
<td>Letter-Quality Printer</td>
</tr>
<tr>
<td>LQP03</td>
<td>Letter-Quality Printer</td>
</tr>
<tr>
<td>LQP15</td>
<td>Letter-Quality Office Printer</td>
</tr>
<tr>
<td>LP25</td>
<td>System Printer</td>
</tr>
<tr>
<td>LP26</td>
<td>System Printer</td>
</tr>
<tr>
<td>LP27</td>
<td>System Printer</td>
</tr>
<tr>
<td>LP29</td>
<td>2000 Line/minute Impact Printer</td>
</tr>
<tr>
<td>LP37</td>
<td>Line Printer</td>
</tr>
<tr>
<td>LVP16</td>
<td>Color Graphics Pen Plotter (C2 configuration only)</td>
</tr>
<tr>
<td>LXY12</td>
<td>Graphics Line Printer</td>
</tr>
<tr>
<td>LXY22</td>
<td>Graphics Line Printer</td>
</tr>
</tbody>
</table>
### A.9 Miscellaneous

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR11</td>
<td>Card reader (C2 configuration only).</td>
</tr>
<tr>
<td>DSS1</td>
<td>Digital Storage Systems Interconnect.</td>
</tr>
<tr>
<td>FP730</td>
<td>Floating Point Accelerator.</td>
</tr>
<tr>
<td>FP750</td>
<td>Floating Point Accelerator.</td>
</tr>
<tr>
<td>FP780</td>
<td>Floating Point Accelerator.</td>
</tr>
<tr>
<td>FP785</td>
<td>Floating Point Accelerator.</td>
</tr>
<tr>
<td>FP86-AA</td>
<td>Floating Point Accelerator.</td>
</tr>
<tr>
<td>FV64A</td>
<td>Vector Processing option.</td>
</tr>
<tr>
<td>H7112</td>
<td>Memory battery back-up.</td>
</tr>
<tr>
<td>KE780</td>
<td>G and H floating point microcode for the VAX-11/780 system.</td>
</tr>
<tr>
<td>SBI</td>
<td>System Backplane Interconnect.</td>
</tr>
<tr>
<td>SCSI</td>
<td>Small Computer System Interface.</td>
</tr>
<tr>
<td>VS40X</td>
<td>4-plane graphics coprocessor.</td>
</tr>
<tr>
<td>WS01X</td>
<td>VAXstation 3100 SPX Graphics option.</td>
</tr>
</tbody>
</table>
This page intentionally left blank
Appendix B

Evaluated Software List (VAX and Alpha)

Evaluated Software for OpenVMS:

- OpenVMS VAX Version 6.1
- OpenVMS Alpha Version 6.1 with ALPRAMP01.061
- HSC Software Version 6.5 for HSC40, 60, 70, 90
- HSC Software Version 4.00 for HSC50
- OpenVMS Extended Documentation Set

Optional system integrated products that may be used in the evaluated configuration:

- VMSccluster (common environment)
- DECnet (within a VMSccluster)
- VMS Volume Shadowing
- RMS Journaling

Optional system integrated products that must be not be used in the evaluated configuration:

- DECwindows
- Local Area Transport (LAT)

Evaluated Software for Security Enhanced VMS (SEVMS):

- OpenVMS evaluated software
- Digital Security Consulting Services\(^1\)
  - SEVMS VAX Version 6.1
  - SEVMS Alpha Version 6.1 with ALPRAMP01.061

---

\(^1\) These consulting services were not considered in the evaluation.
Appendix C

TCB Images (VAX and Alpha)

C.1 VMS Executive Images

<table>
<thead>
<tr>
<th>Image</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLUSTRLOA</td>
<td>VMScluster support(^2) (VAX only)</td>
</tr>
<tr>
<td>CPULOA</td>
<td>Tables of CPU data</td>
</tr>
<tr>
<td>DDIF#RMS_EXTENSION</td>
<td>Support for DDIF file operations</td>
</tr>
<tr>
<td>ERRORLOG</td>
<td>Error logging routines and system services</td>
</tr>
<tr>
<td>EVENT_FLAGS_AND_ASTS</td>
<td>Event flag and AST routines and system services (VAX only)</td>
</tr>
<tr>
<td>EXEC_INIT</td>
<td>Routines required for executive initialization</td>
</tr>
<tr>
<td>EXCEPTION</td>
<td>Exception service routines and system services and bugcheck routines</td>
</tr>
<tr>
<td>F11BXP</td>
<td>Files 11 Level 2 file system</td>
</tr>
<tr>
<td>FPEMUL</td>
<td>Floating point emulation routines(^2) (VAX only)</td>
</tr>
<tr>
<td>IMAGE_MANAGEMENT</td>
<td>Image activation services and routines</td>
</tr>
<tr>
<td>IO_ROUTINES</td>
<td>I/O-related routines and system services</td>
</tr>
<tr>
<td>LMF#GROUP_TABLE</td>
<td>Tables of license data</td>
</tr>
<tr>
<td>LOCKING</td>
<td>Lock management routines and system services</td>
</tr>
<tr>
<td>LOGICAL_NAMES</td>
<td>Logical name routines and system services</td>
</tr>
<tr>
<td>MESSAGE_ROUTINES</td>
<td>Message routines and system services</td>
</tr>
<tr>
<td>PAGE_MANAGEMENT</td>
<td>Page fault service routine, related routines, virtual address space system services (VAX only)</td>
</tr>
<tr>
<td>PRIMITIVE_IO</td>
<td>Primitive console I/O and file system routines (VAX only)</td>
</tr>
<tr>
<td>PROCESS_MANAGEMENT</td>
<td>Scheduling routines and process creation and control system services</td>
</tr>
<tr>
<td>RECOVERY_UNIT_SERVICES</td>
<td>RMS recovery services</td>
</tr>
<tr>
<td>RMS</td>
<td>Record Management Services</td>
</tr>
<tr>
<td>SECURITY</td>
<td>Security-related routines and system services</td>
</tr>
<tr>
<td>SHELLxxxk</td>
<td>Process PHD and PI space shell (Alpha only)</td>
</tr>
<tr>
<td>STACONFIG</td>
<td>MSCP controller configurator during boot</td>
</tr>
</tbody>
</table>
### APPENDIX C. TCB IMAGES (VAX AND ALPHA)

<table>
<thead>
<tr>
<th>Image</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>System Kernel</td>
</tr>
<tr>
<td>SYS$SCS</td>
<td>System Communication Services support</td>
</tr>
<tr>
<td>SYS$CLUSTER</td>
<td>Cluster routines</td>
</tr>
<tr>
<td>SYSDEVICE</td>
<td>Pseudo device drivers and mailbox system services</td>
</tr>
<tr>
<td>SYSGETSYI</td>
<td>Get system information system service</td>
</tr>
<tr>
<td>SYSLOAxxx</td>
<td>CPU model specific code</td>
</tr>
<tr>
<td>SYSLDR_DYN</td>
<td>System image loader for dynamic loading</td>
</tr>
<tr>
<td>SYSLICENSE</td>
<td>License system service</td>
</tr>
<tr>
<td>SYSMSG</td>
<td>System Messages (VAX only)</td>
</tr>
<tr>
<td>SYSTEM_DEBUG</td>
<td>Kernel mode debugger (optional)</td>
</tr>
<tr>
<td>SYSTEM_PRIMITIVES</td>
<td>Basic system support routines</td>
</tr>
<tr>
<td>SYSTEM_SYNCHRONIZATION</td>
<td>Symmetric Multiprocessor Support and error checking</td>
</tr>
<tr>
<td>SYSTEM_SYNCHRONIZATION_MIN</td>
<td>Symmetric Multiprocessor Support</td>
</tr>
<tr>
<td>SYSTEM_SYNCHRONIZATION_UIM</td>
<td>Uniprocessor support</td>
</tr>
<tr>
<td>VAXEMUL</td>
<td>Emulation for extended instruction set for microVAX</td>
</tr>
<tr>
<td>WORKING_SET_MANAGEMENT</td>
<td>Swapper and supporting routines, related system services</td>
</tr>
<tr>
<td>VECTOR_PROCESSING</td>
<td>Vector-processing exceptions and support routines (VAX only)</td>
</tr>
<tr>
<td>VVIEF_BOOTSTRAP</td>
<td>Vector instruction emulation</td>
</tr>
<tr>
<td>xXDRIVER</td>
<td>Various device drivers</td>
</tr>
</tbody>
</table>

---

1 There exists a module for each CPU model.
2 Exactly one of these modules is loaded based on configuration and system parameters.
3 Loaded only on some CPU models.
4 Loaded only if associated hardware is present.
5 Loaded only on VM$cluster nodes.
## C.2 Privileged System Processes

<table>
<thead>
<tr>
<th>Process Name</th>
<th>Image Name</th>
<th>UIC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIT_SERVER</td>
<td>AUDIT_SERVER</td>
<td>[1,4]</td>
<td>Audit server process</td>
</tr>
<tr>
<td>BADBLOCK.Scan</td>
<td>BADBLOCK</td>
<td>[1,4]</td>
<td>Dynamic Bad Block Analysis Facility</td>
</tr>
<tr>
<td>CACHE_SERVER</td>
<td>FILESERV</td>
<td>[1,4]</td>
<td>Cluster disk cache server¹¹</td>
</tr>
<tr>
<td>CLUSTER_SERVER</td>
<td>CSP</td>
<td>[1,4]</td>
<td>Cluster Server Process¹¹</td>
</tr>
<tr>
<td>CONFIGURE</td>
<td>CONFIGURE</td>
<td>[1,4]</td>
<td>Configure Cluster Devices Process¹¹</td>
</tr>
<tr>
<td>ERRFMT</td>
<td>ERRFMT</td>
<td>[1,6]</td>
<td>Error Log Format Process</td>
</tr>
<tr>
<td>EVENT_SERVER</td>
<td>ESP</td>
<td>[1,4]</td>
<td>DEEvent (Alpha only)</td>
</tr>
<tr>
<td>INPSMB</td>
<td>INPSMB</td>
<td>[1,4]</td>
<td>Card Reader Input Symbiont (VAX only)</td>
</tr>
<tr>
<td>IPCACP</td>
<td>IPCACP</td>
<td>[1,4]</td>
<td>$IPC system call server</td>
</tr>
<tr>
<td>JOB CONTROL</td>
<td>JBC$JOB,CONTROL</td>
<td>[1,4]</td>
<td>Job Controller</td>
</tr>
<tr>
<td>MTAACP</td>
<td>MTAACP</td>
<td>[1,3]</td>
<td>Magnetic Tape ACP</td>
</tr>
<tr>
<td>SEVMS$NOLABELMTAAACP</td>
<td>SEVMS</td>
<td>[1,3]</td>
<td>SEVMS Magnetic Tape ACP for unlabeled data</td>
</tr>
<tr>
<td>NETACP</td>
<td>NETACP</td>
<td>[1,3]</td>
<td>Network Control ACP¹¹</td>
</tr>
<tr>
<td>OPCOM</td>
<td>OPCOM</td>
<td>[1,4]</td>
<td>Operator Communication Facility</td>
</tr>
<tr>
<td>REMACP</td>
<td>REMACP</td>
<td>[1,3]</td>
<td>Remote Terminal ACP¹¹</td>
</tr>
<tr>
<td>SECURITY_SERVER</td>
<td>SECURITY_SERVER</td>
<td>[1,4]</td>
<td>Proxy and Intrusion Services (VAX only)</td>
</tr>
<tr>
<td>SHADOW_SERVER</td>
<td>SHADOW_SERVER</td>
<td>[1,4]</td>
<td>Server to copy disks during shadow mounts</td>
</tr>
<tr>
<td>SMISERVER</td>
<td>SMISERVER</td>
<td>[1,4]</td>
<td>VMS System Management Facility¹¹</td>
</tr>
<tr>
<td>SWAPPER</td>
<td></td>
<td>[1,3]</td>
<td>Process Swapper</td>
</tr>
<tr>
<td>SYM.xxx</td>
<td>PRTSMB</td>
<td>[1,4]</td>
<td>Print Symbiont</td>
</tr>
<tr>
<td>SYM.xxx</td>
<td>SEVMS$$SMB</td>
<td>[1,4]</td>
<td>Secure Print Symbiont¹⁰</td>
</tr>
<tr>
<td>TP_SERVER</td>
<td>TPSERV</td>
<td>[1,4]</td>
<td>Transaction Processing Server</td>
</tr>
</tbody>
</table>

¹¹ The process User Identification Code (UIC) is expressed in the format [group number, user number].

¹° The UIC of the SYSTEM account is [1,4].

¹⁰ Used only by SEVMS.

¹¹ Only run in a cluster environment.
C.3 Images Installed with Security Relevant Privileges

<table>
<thead>
<tr>
<th>Image Name</th>
<th>Privileges</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALMDMP</td>
<td>CMEXEC, CMKRN</td>
<td>Image Dump Analysis Utility</td>
</tr>
<tr>
<td>AUTHORIZE</td>
<td>AUDIT</td>
<td>User database maintenance utility</td>
</tr>
<tr>
<td>BACKUP</td>
<td>DOWNGRADE, UPGRADE</td>
<td>Backup and Restore Utility,SEVMS only</td>
</tr>
<tr>
<td>CDU</td>
<td>CMEXEC</td>
<td>Command Definition Utility</td>
</tr>
<tr>
<td>CIA</td>
<td>CMEXEC, AUDIT</td>
<td>Intrusion Detection Utility</td>
</tr>
<tr>
<td>DBUGSSISHR</td>
<td>PROTECTED(^{12})</td>
<td>Debugger system service intercept</td>
</tr>
<tr>
<td>DISMNTSHR</td>
<td>PROTECTED(^{12})</td>
<td>Dismount volume utility</td>
</tr>
<tr>
<td>DTSHSHARE</td>
<td>PROTECTED(^{12})</td>
<td>Distributed transaction services</td>
</tr>
<tr>
<td>EVL</td>
<td>OPER, SYSNAM, SYSPRV</td>
<td>Network Event Logger</td>
</tr>
<tr>
<td>INITSHR</td>
<td>PROTECTED(^{12})</td>
<td>Initialize volume utility</td>
</tr>
<tr>
<td>INSTALL</td>
<td>CMKRN, SYSGBL, PRMGBL, SHMEM, AUDIT</td>
<td>Known Image Installation Utility</td>
</tr>
<tr>
<td>IPCSHR</td>
<td>PROTECTED</td>
<td>$IPC system services</td>
</tr>
<tr>
<td>LOGINOUT</td>
<td>CMKRN, LOG_IO, SYSPRV, SYSNAM, ALTPRI, AUDIT</td>
<td>Login/Logout Image</td>
</tr>
<tr>
<td>MAIL</td>
<td>DOWNGRADE</td>
<td>Mail Utility(^{12}), SEVMS only</td>
</tr>
<tr>
<td>MAIL_SERVER</td>
<td>SYSPRV</td>
<td>Network Mail Utility Server</td>
</tr>
<tr>
<td>MAILSRP</td>
<td>PROTECTED(^{12})</td>
<td>E-Mail services</td>
</tr>
<tr>
<td>MESH</td>
<td>PROTECTED</td>
<td>Media Management Extensions</td>
</tr>
<tr>
<td>MOUNTSHR</td>
<td>PROTECTED(^{12})</td>
<td>Mount volumes on devices utility</td>
</tr>
<tr>
<td>NCP</td>
<td>CMKRN, AUDIT</td>
<td>Network Control Utility</td>
</tr>
<tr>
<td>NICONFIG</td>
<td>LOG_IO, SYSNAM</td>
<td>Ethernet module configurator</td>
</tr>
<tr>
<td>PHONE</td>
<td>OPER, PRMMBX, SYSNAM, WORLD</td>
<td>Phone Utility</td>
</tr>
<tr>
<td>PTDSSERVICES_SHR</td>
<td>PROTECTED(^{12})</td>
<td>Pseudo Terminal driver services</td>
</tr>
<tr>
<td>REPLY</td>
<td>AUDIT</td>
<td>For DCL reply</td>
</tr>
<tr>
<td>SECURESHRP</td>
<td>PROTECTED(^{12})</td>
<td>System security services</td>
</tr>
<tr>
<td>SET</td>
<td>CMKRN, SYSPRV, AUDIT</td>
<td>Set Command Processor for System Space</td>
</tr>
<tr>
<td>SETAUDIT</td>
<td>ALTPRI, BYPASS, CMKRN, DETACH, PSWAPM, SETPR, WORLD, AUDIT</td>
<td>Audit Control Command Processor</td>
</tr>
</tbody>
</table>
### C.3. IMAGES INSTALLED WITH SECURITY RELEVANT PRIVILEGES

<table>
<thead>
<tr>
<th>Image Name</th>
<th>Privileges</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETP0</td>
<td>CMKRNL, SYSPRV, DOWNGRADE, AUDIT</td>
<td>Set Command Processor for Process Space (DOWNGRADE on SEVMS only)</td>
</tr>
<tr>
<td>SETRIGHTS</td>
<td>CMKRNL, AUDIT</td>
<td>Set rightslist command processor</td>
</tr>
<tr>
<td>SETSHOCCLASS</td>
<td>CMKRNL, OPER, AUDIT</td>
<td>Set and Show Classifications(^\text{13}) (SEVMS only)</td>
</tr>
<tr>
<td>SEVMS$SETSHOWAUDIT</td>
<td>CMKRNL</td>
<td>SEVMS Audit Control Command Processor(^\text{13})</td>
</tr>
<tr>
<td>SEVMS$SETSHOWTEMPLATE</td>
<td>CMKRNL</td>
<td>Set and Show the SEVMS Printer Label Templates(^\text{13}) (SEVMS only)</td>
</tr>
<tr>
<td>SHOW</td>
<td>CMKRNL, SYSPRV, WORLD, AUDIT</td>
<td>Show Command Processor</td>
</tr>
<tr>
<td>SMPUTIL</td>
<td>CMKRNL, AUDIT</td>
<td>Show Cluster Command Processor</td>
</tr>
<tr>
<td>SPISHR</td>
<td>PROTECTED(^\text{12})</td>
<td>System Performance Information services</td>
</tr>
<tr>
<td>SYSGEN</td>
<td>AUDIT</td>
<td>System Configuration Utility</td>
</tr>
<tr>
<td>SYSMAN</td>
<td>AUDIT, CMKRNL</td>
<td>System Management Utility</td>
</tr>
<tr>
<td>SYSSSSISHR</td>
<td>PROTECTED</td>
<td>System Services Intercept Support</td>
</tr>
<tr>
<td>TFU</td>
<td>CMKRNL</td>
<td>Terminal Fallback Utility (VAX only)</td>
</tr>
<tr>
<td>VPM</td>
<td>ALTPRI, PSWAPM, SYSNAM, SYSPRV</td>
<td>System Performance Monitoring Tool</td>
</tr>
</tbody>
</table>

\(^{12}\) Images listed with the privilege “PROTECTED” are installed with the `/PROTECT` option and run in the protected modes of the processor.

\(^{13}\) Installed only on SEVMS.
Appendix D

Acronyms

ACB       AST Control Block
ACE       Access Control Entry
ACL       Access Control List
ACP       Ancillary Control Processes
ANSI      American National Standards Institute
ARB       Access Rights Block
AST       Asynchronous System Trap
AXE       Architecture Exerciser
BI        Backplane Interconnect
BICPSW    Bit Clear PSW
BISPSW    Bit Set PSW
CCB       Channel Control Block
CEB       Common Event Block
CEF       Common Event Flag
CEFC      Common Event Flag Cluster
CHME      Change Mode to Executive
CHMK      Change Mode to Kernel
CHMS      Change Mode to Supervisor
CHMU      Change Mode to User
CI        Computer Interconnect
CLI       Command Language Interpreter
CM        Compatibility Mode
CMI       CPU-to-Memory Interconnect
CPU       Central Processing Unit
CISC      Complex Instruction Set Computing
CSR       Control and Status Register
DAC       Discretionary Access Control
DAP       Design Analysis Phase
DAP       Data Access Protocol
DCL       Digital Command Language
DDI       DR32 Device Interconnect
DDT       Driver Dispatch Table
APPENDIX D. ACRONYMS

DEC  Digital Equipment Corporation
DELNI  Device Local Network Interconnect
DMA  Direct Memory Access
DSA  Digital Storage Architecture
DSSI  Digital Storage Systems Interconnect
DTM  Digital Test Manager
ECC  Error Correcting Code
EPL  Evaluated Products List
ERF  Error Log Report Formatter
FAB  File Access Block
FAL  File Access Listener
FER  Final Evaluation Report
FCB  File Control Block
FDDI  Fiber Distributed Data Interface
FDL  File Definition Language
FDT  Function Decision Table
FPD  First Part Done
GID  Group Identifier
GPT  Global Page Table
GPTE  Global Page Table Entry
GPTX  Global Page Table Index
GPX  Graphics Processing Accelerator
GSD  Global Section Descriptor
GSTE  Global Section Table Entry
HSC  Hierarchical Storage Controller
IHD  Image Header
IOP  I/O Port Module
IPC  Interprocess Communication
IPL  Interrupt Priority Level
IPR  Internal Processor Register
IRP  I/O Request Packet
IS  Interrupt Stack
ISA  Integrated Storage Assembly
ISO  International Organization for Standardization
JIB  Job Information Block
KFE  Known File Entry
KRP  Kernel Request Packet
LAN  Local Area Network
LAT  Local Area Transport
LAVc  Local Area VAXcluster
LDPCTX Load Process Context
LKB  Lock Block
LSB  Laser System Bus
MAC  Mandatory Access Control
MAX  Multi-instruction Architecture Exerciser
MBA  MASSBUS Adapter
MFB  Monochrome Frame Buffer
MFD  Master File Directory
MBZ  Must Be Zero
MFPR Move From Privileged Register
MFVP  Move From Vector Processor
MI  Mixed-Interconnect
MOP  Maintenance Operations Protocol
MSCP Mass Storage Control Protocol
MTPR Move To Privileged Register
MTVP Move To Vector Processor
MVCF  Military VAX Computer Family
NAM  Name Block
NCP  Network Control Program
NCSC National Computer Security Center
NI  Network Interconnect
NSA  National Security Agency
ODS  On-Line Disk Structure
OCB  Object Class Block
OPCOM  Operator Communication Manager
ORB  Object Rights Block
PC  Program Counter
PCB  Process Control Block
PDP  Programmable Data Processor
PFN  Page Frame Number
PHD  Process Header
PID  Process ID
PQB  Process Quota Block
PS  Processor Status
PSL  Processor Status Longword
PST  Process Section Table
PSTE  Process Section Table Entry
PSTX  Process Section Table Index
PSW  Processor Status Word
## APPENDIX D. ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTE</td>
<td>Page Table Entry</td>
</tr>
<tr>
<td>PTR</td>
<td>Preliminary Technical Review</td>
</tr>
<tr>
<td>QAR</td>
<td>Quality Assurance Report</td>
</tr>
<tr>
<td>QVSS</td>
<td>Q-Bus Video Subsystem</td>
</tr>
<tr>
<td>RAB</td>
<td>Record Access Block</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RAMP</td>
<td>Rating Maintenance Phase</td>
</tr>
<tr>
<td>RAX</td>
<td>RISC/Alpha AXP Architecture Exerciser</td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced Instruction Set Computing</td>
</tr>
<tr>
<td>RCO</td>
<td>Responsible Corporate Officer</td>
</tr>
<tr>
<td>RDAB</td>
<td>Resource Domain Access Block</td>
</tr>
<tr>
<td>RDDB</td>
<td>Resource Domain Data Block</td>
</tr>
<tr>
<td>REI</td>
<td>Return from Exception or Interrupt</td>
</tr>
<tr>
<td>RMS</td>
<td>Record Management Services</td>
</tr>
<tr>
<td>ROM</td>
<td>Read Only Memory</td>
</tr>
<tr>
<td>RSB</td>
<td>Resource Block</td>
</tr>
<tr>
<td>SBI</td>
<td>Synchronous Backplane Interconnect</td>
</tr>
<tr>
<td>SBR</td>
<td>System Base Register</td>
</tr>
<tr>
<td>SCA</td>
<td>Systems Communications Architecture</td>
</tr>
<tr>
<td>SCB</td>
<td>System Control Block</td>
</tr>
<tr>
<td>SCBB</td>
<td>System Control Block Base</td>
</tr>
<tr>
<td>SCS</td>
<td>System Communication Services</td>
</tr>
<tr>
<td>SCSI</td>
<td>Small Computer System Interface</td>
</tr>
<tr>
<td>SCT</td>
<td>Source Change Tracking</td>
</tr>
<tr>
<td>SDI</td>
<td>Standard Disk Interconnect</td>
</tr>
<tr>
<td>SES</td>
<td>Security Enhancement Services</td>
</tr>
<tr>
<td>SEVMS</td>
<td>Security Enhanced VMS</td>
</tr>
<tr>
<td>SFUG</td>
<td>Security Features User’s Guide</td>
</tr>
<tr>
<td>SP</td>
<td>Stack Pointer</td>
</tr>
<tr>
<td>SPD</td>
<td>Software Product Description</td>
</tr>
<tr>
<td>SPT</td>
<td>System Page Table</td>
</tr>
<tr>
<td>SPTE</td>
<td>System Page Table Entry</td>
</tr>
<tr>
<td>SPX</td>
<td>2D Scanline Processor Accelerator Graphics System</td>
</tr>
<tr>
<td>SSRT</td>
<td>Software Security Response Team</td>
</tr>
<tr>
<td>STI</td>
<td>Standard Tape Interconnect</td>
</tr>
<tr>
<td>SVPCTX</td>
<td>Save Process Context</td>
</tr>
<tr>
<td>SYSAP</td>
<td>System Application</td>
</tr>
<tr>
<td>TCB</td>
<td>Trusted Computing Base</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>TFM</td>
<td>Trusted Facility Manual</td>
</tr>
<tr>
<td>TMSCP</td>
<td>Tape Mass Storage Control Protocol</td>
</tr>
<tr>
<td>TRB</td>
<td>Technical Review Board</td>
</tr>
<tr>
<td>UAF</td>
<td>User Authorization File</td>
</tr>
<tr>
<td>UBA</td>
<td>UNIBUS Adapter</td>
</tr>
<tr>
<td>UCB</td>
<td>Unit Control Block</td>
</tr>
<tr>
<td>UETP</td>
<td>User Environment Test Package</td>
</tr>
<tr>
<td>UIC</td>
<td>User Identification Code</td>
</tr>
<tr>
<td>UID</td>
<td>User Identifier</td>
</tr>
<tr>
<td>VAP</td>
<td>Vendor Assistance Phase</td>
</tr>
<tr>
<td>VAX</td>
<td>Virtual Address eXtension</td>
</tr>
<tr>
<td>VBPOC</td>
<td>Vendor Business Point of Contact</td>
</tr>
<tr>
<td>VCB</td>
<td>Volume Control Block</td>
</tr>
<tr>
<td>VP</td>
<td>Vector Processor</td>
</tr>
<tr>
<td>VME</td>
<td>Versabus Module Eurocard</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Page Number</td>
</tr>
<tr>
<td>VSA</td>
<td>Vendor Security Analyst</td>
</tr>
<tr>
<td>WCB</td>
<td>Window Control Block</td>
</tr>
<tr>
<td>WSL</td>
<td>Working Set List</td>
</tr>
<tr>
<td>WSLE</td>
<td>Working Set List Entry</td>
</tr>
<tr>
<td>XAB</td>
<td>Extended Attribute Block</td>
</tr>
<tr>
<td>XFC</td>
<td>Extended Function Code</td>
</tr>
<tr>
<td>XQP</td>
<td>Extended QIO Processor</td>
</tr>
</tbody>
</table>
Appendix E

Bibliography and References


Evaluation Documentation References


[8] OpenVMS/SEVMS Design Documentation Roadmap, RAMP V6.1

OpenVMS User Documentation References


Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha

APPENDIX E. BIBLIOGRAPHY AND REFERENCES

Security Enhanced VMS (SEVMS) Documentation References


Design Documentation References


**Digital Hardware Documentation**


Final Evaluation Report Digital OpenVMS and SEVMS, Version 6.1, with VAX or Alpha

[86] FDDI Primer, Digital Equipment Corporation
[87] Field Service Hardware and Diagnostic MicroFICHE, MD.VAX-01, Digital Equipment Corporation, 1992
[99] Workstation Family Technical Summary, Digital Equipment Corporation

Military VAX Documentation References

[103] MVCF 860 Technical Guide, Raytheon Corporation
[104] MVCF RSCSI Technical Guide, Raytheon Corporation
[105] MVCF REBNA Technical Guide, Raytheon Corporation